CEPIS UPGRADE
The European Journal for the Informatics Professional
http://www.upgrade-cepis.org
Vol. VIII, No. 6, December 2007

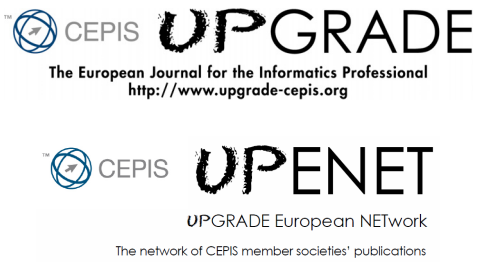
Free Software: Research and Development

http://www.cepis.org

CEPIS, Council of European Professional Informatics Societies, is a non-profit organisation seeking to improve and promote high standards among informatics professionals in recognition of the impact that informatics has on employment, business and society.

CEPIS unites 37 professional informatics societies over 33 European countries, representing more than 400,000 ICT professionals.

CEPIS promotes



http://www.eucip.com http://www.ecdl.com http://www.upgrade-cepis.org

UPGRADE is the European Journal for the Informatics Professional, published bimonthly at <http://www.upgrade-cepis.org/>

Publisher
UPGRADE is published on behalf of CEPIS (Council of European Professional Informatics Societies, <http://www.cepis.org/>) by **Novática** <http://www.ati.es/novatica/>, journal of the Spanish CEPIS society ATI (*Asociación de Técnicos de Informática,* <http://www.ati.es/>)

UPGRADE monographs are also published in Spanish (full version printed; summary, abstracts and some articles online) by **Novática**

UPGRADE was created in October 2000 by CEPIS and was first published by **Novática** and **INFORMATIK/INFORMATIQUE**, bimonthly journal of SVI/FSI (Swiss Federation of Professional Informatics Societies, <http://www.svifsi.ch/>)

UPGRADE is the anchor point for **UP**ENET (**UP**GRADE European NETwork), the network of CEPIS member societies' publications, that currently includes the following ones:
- **Informatik-Spektrum**, journal published by Springer Verlag on behalf of the CEPIS societies GI, Germany, and SI, Switzerland
- **ITNOW**, magazine published by Oxford University Press on behalf of the British CEPIS society BCS
- **Mondo Digitale**, digital journal from the Italian CEPIS society AICA
- **Novática**, journal from the Spanish CEPIS society ATI
- **OCG Journal**, journal from the Austrian CEPIS society OCG
- **Pliroforiki**, journal from the Cyprus CEPIS society CCS
- **Pro Dialog**, journal from the Polish CEPIS society PTI-PIPS

ISSN 1684-5285

Monograph of next issue (February 2008)
"ICT Governance"

(The full schedule of **UP**GRADE is available at our website)

CEPIS **UP**GRADE
The European Journal for the Informatics Professional
http://www.upgrade-cepis.org
Vol. VIIII, issue No. 6, December 2007

Monograph: Free Software: Research and Development
(published jointly with Novática*)
Guest Editors: *Manuel Palomo-Duarte, José-Rafael Rodríguez-Galván, Israel Herraiz-Tabernero, and Andrea Capiluppi*

UPENET (**UP**GRADE European NETwork)

* This monograph will be also published in Spanish (full version printed; summary, abstracts, and some articles online) by **Novática**, journal of the Spanish CEPIS society ATI (*Asociación de Técnicos de Informática*) at <http://www.ati.es/novatica/>.

Presentation

Free Software: Scientific and Technological Innovation

Andrea Capiluppi, José-Rafael Rodríguez-Galván, Manuel Palomo-Duarte, and Israel Herraiz-Tabernero

In recent years we have seen how free software has evolved from being a software development model (with all its ethical and technical implications) to playing a key role in the development strategies of companies, institutions, regions, and even entire countries. Examples such as the Brazilian Government's support of Free Software [1][2] or the Andalusian Regional Government's adoption of free licensing for all its developments [3][4][5], have caused more and more institutions and associations to study the long term implications of adopting the free software model.

One of the most important milestones was the "*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"* [6] developed for the European Commission by UNU-MERIT. It concludes that Free Software offers one of the best chances for the European ICT sector to become a worldwide player and promote RDI (Research, Development & Innovation) initiatives.

In the framework of this scenario we have published this special issue of ***Novática*** and **UP**GRADE on "Free Software: research and development", almost an annual event for the IT community. As usual, most of its content is published under a free license.

After a brief introductory article entitled "*The Need for Libre Software Research in Europe*" by the guest editors of the monograph, we kick off with the paper "*From the Cathedral to the Bazaar: an Empirical Study of the Lifecycle of Volunteer Community Projects*" which presents a comparison between the development communities of two prestigious free software projects, Wine and Arla. In particular the article compares the number of developers who have contributed to the project during its lifecycle.

The Guest Editors

Andrea Capiluppi obtained his Ph.D. from the *Politecnico di Torino*, Italy. In October 2003 he was a visiting researcher in the *Grupo de Sistemas y Comunicaciones* of the *Universidad Rey Juan Carlos*, Madrid, Spain. From January 2004 to the present he has been a visiting researcher in the Department of Maths and Computing at the Open University, UK, working in collaboration with Drs. Juan Ramil, Neil Smith, Helen Sharp, Alvaro Faria, and Sarah Beecham. This appointment has been renewed until December 2008. In January 2006, he joined the University of Lincoln as a Senior Lecturer. <acapiluppi@lincoln.ac.uk>.

José-Rafael Rodríguez-Galván works as a lecturer in the Department of Mathematics at the *Universidad de Cádiz*. Since 2004 he has chaired OSLUCA (Libre Software Office of the *Universidad de Cádiz*), organizing several projects including the 1st, 2nd, and 3rd Free Software Conferences at the *Universidad de Cádiz* and the 1st FLOSS International Conference (FLOSSIC 2007). He has been invited as a speaker to many meetings and symposiums relating to libre software and University. He is also member of UCA researching group FQM-315, where he develops his research in numerical simulation of equations for partial derivatives applied to fluid mechanics. <rafael.rodriguez@uca.es>.

Manuel Palomo-Duarte received his M.Sc. degree in Computer Science from the *Universidad de Sevilla* (2001). He works as a full-time lecturer in the Department of Computer Languages and Systems at the *Universidad de Cádiz* where he teaches subjects related to operating systems and videogame design using libre software. He is also an Erasmus Coordinator for the B.Sc degree in Computer Science "*Ingeniería Técnica en Informática de Sistemas*" He is a member of the "Software Process Improvement and Formal Methods" research group and he is pursuing his Ph.D. on quality in BPEL web services compositions. Since he joined the *Universidad de Cádiz* he has collaborated with the Free Software Office, mainly in relation to the following conferences: 3rd Free Software Conference at the *Universidad de Cádiz* (JOSLUCA3) and the 1st FLOSS International Conference (FLOSSIC 2007). <manuel.palomo@uca.es>.

Israel Herraiz-Tabernero is a Ph.D. student at the *Universidad Rey Juan Carlos,* Madrid, Spain. His research is related to the evolution of libre software projects. In particular, he is using time series analysis and other statistical methods to characterize the evolution of software projects. He has participated in several research projects funded by the Framework Programme of the European Commission (QUALOSS, FLOSSMetrics, QUALIPSO, CALIBRE). He has also collaborated on other projects funded by companies such as Vodafone and Telefonica. He has participated in the writing of manuals about managing and starting libre software projects. For example, together with Juan José Amor and Gregorio Robles he wrote a manual for the *Universitat Oberta de Catalunya's* Master Programme in Free Software. He has been a reviewer for the IEEE Africon 2007 among other conferences and for the journal IEEE Transactions on Software Engineering. He is currently a research and teaching assistant at the *Universidad Rey Juan Carlos*, pursuing his PhD on the evolution of libre software. He also coordinates the programme of the Libre Software Master offered by the *Universidad Rey Juan Carlos*, in collaboration with Igalia and Caixa Nova. <www.herraiz@gsyc.escert.urjc.es>.

Based on these metrics and an analysis of information available from the project (such as ChangeLogs), the author concludes that the cathedral and bazaar models are not mutually exclusive during the lifecycle of a volunteer community project. While remaining in a cathedral phase does not necessarily imply failure (because the project may be meeting its goals), transition to a bazaar model would move the project on to a phase in which the development community would continue to grow. And it is the development community who can make this change happen.

Next up is one of the most interesting articles published in the "Workshop on Emerging Trends in FLOSS Research and Development 2007" (FLOSS 2007) [7], "*The Commons as New Economy and what this Means for Research*". This paper looks at how the ICT world would change if companies were to adopt and develop free software en masse. It analyses some of the consequences, such as a drastic drop in the cost of licenses or the reduction of the risk and cost of software experimentation. This would lead to a really interesting scenario and would open up new avenues in ICT teaching since the latest source code would be available to be studied and improved on by students. Programming would change radically, and it would become a matter of finding and integrating code rather than a creating new code from scratch Also the monetary and human resources needed to develop and deploy Ultra-Large Scale Systems would be reduced.

The paper "*Libre Software for Research*" by the Systems and Communications Group, *Universidad Rey Juan Carlos* (Spain), demonstrates how research groups can benefit from the adoption of a free software methodology. This methodology and its associated protocols can improve communication between globally distributed members and increase the visibility of reports, products, and internal information. All, naturally, in a free software environment.

The next paper is focused on telecommunications: "*Technological Innovation in Mobile Communications Developed with Free Software: Campus Ubicuo".* It describes the results of a collaboration between the GITACA research group and a company supported by the Extremadura regional government (Spain). This project has developed a solution (Campus Ubicuo) for the increasing demand for services and the need for mobility that has changed the traditional model of Internet connectivity based solely on access via fixed networks. Campus Ubicuo has been developed using free/libre software and aims to offer user ubiquity through advanced communications services over wireless networks.

Another paper showing the results of an investment in free software by a public institution is "*The Case of the University of Cádiz's Free Software Office among Spanish Universities*". The paper describes the work done by the Free Software Office of the University of Cadiz (Spain) since it was set up in 2004. One of the most important features of an institution attached to a university is its broad scope of action. Several kinds of initiatives have been developed in the fields of teaching, research, management, support of the development and dissemination of free software, and collaborations with external institutions.

The next paper, also related to RDI and free software, is "*On Understanding how to Introduce an Innovation to an Open Source Project*". Like one of the earlier articles, this paper was first published in FLOSS 2007. It describes a methodology for incorporating software engineering inventions into free software projects. This not only benefits researchers by allowing them to test their tools, methods, and process designs in real-life settings, but it also benefits the free software community by allowing them to apply the latest academic innovations to their projects. But introducing a new artefact into a community which has been working without it for a long time is no simple task. The steps to be taken to ensure successful adoption differ widely depending on the kind of innovation and on the structure and size of the community.

From another Free Software Conference, FLOSSIC 2007 we have selected the paper, "*3D Distributed Rendering and Optimization using Free Software*". This paper received an award as the best paper of the conference. It is the result of a research effort by two European institutions: the *Universidad de Castilla La Mancha* (Spain) and the Software Competence Center at Hagenberg (Austria). The papers deals with a classical computing problem, image generation: in particular how 2D photorealistic images can be obtained from the abstract definition of a 3D scene. The use of free software tools and state-of-the-art distributed techniques and algorithms reduces the computational cost of the process. The free software tools used for distributed rendering optimization in this particular case were Yafrid and MagArRo, both developed at the *Universidad de Castilla-La Mancha*

For our final article we have taken another interesting paper from FLOSS 2007, "*Identifying Success and Tragedy of FLOSS Commons: a Preliminary Classification of Sourceforge.net Projects*" It researches why some free software projects succeed or fail (a tragedy). Although success or failure is very difficult to measure, the authors use collective action (CVS changes, stable versions released in the past year, downloads, etc) as criteria for classifying projects. They develop a different kind of classification of success or tragedy in projects, based on their number of developers, project size, and other metrics.

We would like to conclude our presentation by thanking the staff of ***Novática*** and **UP**GRADE for entrusting us with this special issue. And, of course, we would like to thank everyone whose work has contributed to the publication of this issue: authors, reviewers, translators and, in general, the whole community that makes Free Software and Knowledge a reality.

Useful References on Free Software

The following references, along with those included in the articles this monograph consists of, will help our readers to dig deeper into this field.

[1] <http://www.nytimes.com/2005/03/29/technology/29computer.html>.

[2] <http://news.bbc.co.uk/1/hi/business/4602325.stm>.

[3] Decree 72/2003 on Measures for Advancing the Knowledge Society in Andalusia, of March 18, 2003 (BOJA 55, March 21, 2003)

[4] <http://www.20minutos.es/noticia/91463/0/programas/ordenador/pueden/>.

[5] <http://www.juntadeandalucia.es/repositorio/>.

[6] <http://ec.europa.eu/enterprise/ict/policy/doc/2006-11-20-flossimpact.pdf>.

[7] <http://cross.lincoln.ac.uk/floss2007/>.

Institutions Supporting Free Software

- Free Software Foundation <http://fsf.org>.
- Open Source Initiative <http://opensource.org>.
- Cenatic <http://www.cenatic.es/>.
- OSLUCA <http://www.uca.es/softwarelibre>.

News Sites

- **Slashdot** <ttp://slashdot.org>.
- **Digg** <ttp://digg.com>.
- **Blog de Ricardo Galli** (in Spanish) <http://ricardogalli.com>.
- **Meneame** (in Spanish) <http://meneame.net>.
- **Barrapunto** (in Spanish) <http://barrapunto.com>.

Books

- Eric S. Raymond. The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary, O'Reilly, 2001, ISBN: 0596001088. http://www.catb.org/~esr/writings/cathedral-bazaar/.
- Richard M. Stallman, Lawrence Lessig, and Joshua Gay (Editor). Free Software, Free Society: Selected Essays of Richard M. Stallman, Free Software Foundation, 2002, ISBN: 1-882114-98-1. <http://www.gnu.org/philosophy/fsfs/rms-essays.pdf>.
- Lawrence Lessig. Code 2.0. Basic Books, 2006. ISBN-13: 978–0–465–03914–2. <http://codev2.cc/>.
- Eric Von Hippel. Democratizing Innovation. MIT Press, 2006. ISBN-13: 9780262002745. <http://web.mit.edu/evhippel/www/democ1.htm>.
- Ron Goldman and Richard P. Gabriel. Innovation Happens Elsewhere: Open Source as Business Strategy. Morgan Kaufman/Elsevier, 2005, ISBN: 1-55860-889-3. <http://dreamsongs.com/IHE/>.
- Peter Wayner. Free for All: How Linux and the Free Software Movement Undercut the High-Tech Titans. Peter Wayner , 2000. ISBN 0-06-662050-3. <http://www.rau-tu.unicamp.br/nou-rau/software-livre/document/?code=138>.
- O'Reilly Open Books project. <http://www.oreilly.com/openbook/>.
- Lawrence Rosen. Open Source Licensing: Software Freedom and Intellectual Property Law. Prentice Hall, 2004. ISBN-13: 978-0131487871. <http://www.rosenlaw.com/oslbook.htm>.
- Joseph Feller, Brian Fitzgerald, Scott A. Hissam, and Karim R. Lakhani. Perspectives on Free and Open Source Software. ISBN-13: 978-0-262-06246-6. MIT Press, 2006. <http://mitpress.mit.edu/catalog/item/default.asp?tid=10477&ttype=2>.
- Karl Fogel. Producing Open Source Software: How to Run a Successful Free Software Project. Karl Fogel, 2005. <http://producingoss.com/>.
- Linux Torvalds and David Diamond. Just for Fun, The story of an accidental revolutionary. HarperCollins, 2001. ISBN-13: 978-0066620725.
- Glyn Moody. Rebel Code: Linux and the Open Source Revolution. Perseus Books Group, 2002. ISBN-13: 978-0738206707
- A. Abella, M. A. Segovia. White book on Free Software in Spain (in Spanish). 2007. <http://www.libroblanco.com>.

Other Interesting Links

- Economic and Game Theory: Against Intellectual Monopoly. <http://levine.sscnet.ucla.edu/general/intellectual/againstnew.htm>.
- FLOSSIC 2007 Free documentation compilation. <http://flossic.loba.es/>.
- Free Resources created for Free Software postgraduated courses at UOC (in Spanish). <http://www.uoc.edu/masters/esp/web/materiales_libres.html>.
- European Interoperability Framework for pan-European eGovernment Services. European Communities, 2004. ISBN 92-894-8389-X. <http://europa.eu.int/idabc/en/document/3761>.
- How to Collaborate with the KDE project (in Spanish). <http://www.kdehispano.org/colaborar_KDE>.
- Debian project. <http://www.debian.org>.
- Guiactiva: guide to creating Free Software Companies (in Spanish). CEIN, S.A., 2005. Legal deposit no.: NA 1078-2005. <http://www.cein.es/web/es/documentacion/ideas/2005/7831.php>.
- Linux Knowledge Base and Tutorial. <http://sourceforge.net/projects/linkbat>.
- Mitsubishi Research Institute, Inc. An Introduction to Open Source Software. 2006. <http://oss.mri.co.jp/i2oss/download/en/text.pdf>.
- Alessio Damato. Why The Future Of Science Must Be In Free Software. <http://scientificcomputing.net/debian/why.pdf>.

The Need for Libre Software Research in Europe

Israel Herraiz-Tabernero, José-Rafael Rodríguez-Galván, and Manuel Palomo-Duarte

The European Commission, by means of the Framework Programme, is funding several research projects on libre software. In the sixth edition of this programme, the sum of 25.13 million Euros has been dedicated to fund these research projects. Is this investment worthwhile? Can libre software help the development of Europe? In this editorial, we expose the reasons that justify this research , and how the research projects can foster the social and economic development of Europe. Finally, we include a summary of the main research projects funded in the scope of the Framework Programme.

Keywords*:* Framework Programme, Research.

1 Libre Software in Europe

Libre software was born as a result of communities of volunteers, joining forces to develop software. The main motivations to work in the projects were personal. However, libre is now recognised as an important economic phenomenon causing the giants of software to adopt a strategy towards libre software: some regard it as a threat to their businesses, while others see it as an opportunity to open new markets and reinforce their competitiveness. Libre software respects standards, allows interoperability among (and within) public and private institutions and avoids monopolies in the access to information. It also reinforces a neutral education, insome cases supplanting the commercial products used for educational purposes. Thus, universities and research centres, that were involved in the libre software movement since its beginning (in parallel with the development of the internet), are focusing on libre software, by promoting it for the daily tasks of these centres (research, education, management, etc), and by studying it as a matter of research. Its study is becoming more important because software is a fundamental agent in the economy. It is present everywhere: in offices, in mobile phones, in cars, in the public systems that manage our

Authors

Israel Herraiz-Tabernero is a Ph.D. student at the *Universidad Rey Juan Carlos,* Madrid, Spain. His research is related to the evolution of libre software projects. In particular, he is using time series analysis and other statistical methods to characterize the evolution of software projects. He has participated in several research projects funded by the Framework Programme of the European Commission (QUALOSS, FLOSSMetrics, QUALIPSO, CALIBRE). He has also collaborated on other projects funded by companies such as Vodafone and Telefonica. He has participated in the writing of manuals about managing and starting libre software projects. For example, together with Juan José Amor and Gregorio Robles he wrote a manual for the *Universitat Oberta de Catalunya's* Master Programme in Free Software. He has been a reviewer for the IEEE Africon 2007 among other conferences and for the journal IEEE Transactions on Software Engineering. He is currently a research and teaching assistant at the *Universidad Rey Juan Carlos*, pursuing his PhD on the evolution of libre software. He also coordinates the programme of the Libre Software Master offered by the *Universidad Rey Juan Carlos*, in collaboration with Igalia and Caixa Nova. <www.herraiz@gsyc.escert.urjc.es>.

José-Rafael Rodríguez-Galván works as a lecturer in the Department of Mathematics at the *Universidad de Cádiz*. Since 2004 he has chaired OSLUCA (Libre Software Office of the *Universidad de Cádiz*), organizing several projects including the 1st, 2nd, and 3rd Free Software Conferences at the *Universidad de Cádiz* and the 1st FLOSS International Conference (FLOSSIC 2007). He has been invited as a speaker to many meetings and symposiums relating to libre software and University. He is also member of UCA researching group FQM-315, where he develops his research in numerical simulation of equations for partial derivatives applied to fluid mechanics. <rafael.rodriguez@uca.es>.

Manuel Palomo-Duarte received his M.Sc. degree in Computer Science from the *Universidad de Sevilla* (2001). He works as a full-time lecturer in the Department of Computer Languages and Systems at the *Universidad de Cádiz* where he teaches subjects related to operating systems and videogame design using libre software. He is also an Erasmus Coordinator for the B.Sc degree in Computer Science "*Ingeniería Técnica en Informática de Sistemas*" He is a member of the "SoftwareProcess Improvement and Formal Methods" research group and he is pursuing his Ph.D. on quality in BPEL web services compositions. Since he joined the *Universidad de Cádiz* he has collaborated with the Free Software Office, mainly in relation to the following conferences: 3rd Free Software Conference at the *Universidad de Cádiz* (JOSLUCA3) and the 1st FLOSS International Conference (FLOSSIC 2007). <manuel.palomo@uca.es>.

personal data, etc. However, there is not yet a true Engineering that provides the tools to build it. It is obvious that Software Engineering does exist, but the Brook's *Mythical Man Month* [1] is still present: *We cannot predict how long a software project will take, exactly what the final features will be, what defects will be present or how much money will finally have to be invested. Furthermore, the results of long projects are usually complex products, difficult to maintain. Sometimes, it is even better to start a new project from scratch rather than use a previous existing product as a base*. The software that is the main trunk of the current economic system is not made out in our economic frontiers: is not written by European engineers, neither it is sold by European companies. We could therefore say that a crucial industry for the European economy is not in the hands of Europe. It is also recognised that Software Engineering is not a truly scientific discipline. Many research paper authors have to sign non-disclosure agreements in order to gain access to data sources. In some cases, it is not even known which are the case studies included in the paper. These case studies are often labelled under obscure names such as A,BC or X,Y, etc. Thus, repetition and verification of the results is impossible. We will never achieve equality with the development stage of other scientific disciplines with all these obstacles. We will never overcome the software crisis using a scientific discipline that discourages innovation by means of obstacles (such as non-disclosure agreements). Libre software can help to overcome all these difficulties. First of all, regarding economic impact, libre software is controlled by no one (or,from other point of view, it is controlled by everyone). Currently, libre software has an important impact in the European economy. The recently published report *The impact of Free/Libre/Open Source Software on innovation and competitiveness of the European Union* [2] mentions that 20% of the European investment on software is made on libre software (this amount is similar for the USA), and that in 2010, the global impact of libre software will account for 4% of the GID. Because of its characteristics, from the research point of view libre software does not impose any obstacle to the advancement of science, because all the data sources are public. Furthermore, the report mentioned in the above paragraph also states that many libre products are market leaders in their niches. In other words, those products present a quality level that is good enough to overtake other solutions that have been developed in-house in industrial environments. This means that in spite of the lack of a scientific base on how we develop software, quality products are being produced. What is even better is that the development process in libre software leaves some trails (documentation, source code, change log records, e-mail communications, etc), and all those trails are publicly available. Therefore, libre software constitutes a true research laboratory in studying how to overcome the problems addressed by Brooks and which we are still facing. In this sense, in the scope of the 6th Framework Programme (6FP), 11 research projects were launched, with a global budget of more than 25 million Euros (see Table 1). Within the 7th edition of the Framework Programme, some additional research projects on the same topic are being funded as well. Focusing on the 6FP, the description and goals of the projects vary:

- **Qualipso**

This is the largest research project on libre software that has been funded by the European Commission. The first year of this project has resulted in the celebration of the Qualipso conference in January 2008 in Rome (Italy), with companies from all over Europe participating.
The main goal of Qualipso is to define and implement the technologies, processes and policies to facilitate the development and use of libre software components, with the same level of trust traditionally offered by proprietary software.

- **TOSSAD**

TOSSAD is a coordinated activity with the purpose of diffusing libre software in the public and private sectors, and hopes to create a consortium for this purpose. It will try to identify synergies in order to foster innovation by adopting libre software.

- **SELF**

This project is similar to TOSSAD, but it is more focused on educational resources. The main idea is to take advantage of libre software features (availability of resources, provider independence, low cost of licenses, etc) in the educational sector.

- **FLOSSWorld**

This project studied the situation of libre software in the different world regions: Europe, Asia, Africa, North America and Latin America. The main goal was to make sure that Europe leads research on open source in the world, as well as determine the current situation regarding libre software development, industry, standards, interoperability and e-government in the different world regions.

- **FLOSSMetrics**

This project is collecting metrics and information about a large set of libre software projects (in the order of thousands). The goal is to create a database that could be used by third parties such as researchers, companies and even libre software projects themselves. This project is trying to coordinate with other projects from the 6FP,that need to collect metrics for their particular purposes (for instance, there is a close collaboration with the QUALOSS project).

- **TEAM**

This project is developing a knowledge sharing environment, based on libre software. The final system will be released as libre software as well.

- **EDOS**

Libre software distributions (such as Red Hat, Debian, Ubuntu, Suse, etc) face many common problems. Most of the problems are related to dependencies among the different packages of the distribution. Even today it is common to crash a running system by installing an upgrade of

the packages, because of this dependencies issue. Furthermore, the development and maintenance of those distributions is becoming more and more complex because of the growing interactions among packages (these interactions grow with the square of the number of packages). In order to address these problems, this project tried to provide tools to manage packages installation and distributions maintenance.

- **CALIBRE**

This project tried to be a meeting point between secondary sector companies i.e. those companies that do not develop software but whose businesses crucially depend on software. As a result of this project, an industrial forum was created, called CALIBRATION, which many European companies belong to (Philips, Telefónica and Vodafone among others).

- **SQO-OSS**

This project is trying to develop a model for the evaluation of the quality of libre software, by means of empirical methods. It uses the public data sources of some libre software projects. Among these sources, we may find source code version control repositories, mailing list archives, bug tracking systems, source code, etc. Its main aim is very close to QUALOSS.

- **PYPY**

With the goal of porting Python (a well known programming language) to more platforms, thereby making it more flexible for adaptation to new systems, this project is creating a new implementation of Python. An interesting point of this project is that it is using agile methods in the software development tasks, and the consortium is organized following the schemes of a libre software community. All the development process is being monitored, with the goal of making research the impact of agile software development in the development process.

- **QUALOSS**

The main goal of QUALOSS is to create an evaluation model for the quality of libre software projects. For this purpose, several publicly available data sources are being used. This project will study 50 different software projects, and will try, wherever possible, to reuse the information and databases provided by FLOSSMetrics.

Project	Start date	Duration (months)	EC budget (m€)	Total budget (m€)
Qualipso	Nov-06	48	10.42	17.29
TOSSAD	Feb-05	25	0.78	0.79
SELF	July-06	24	0.98	0.98
FLOSSWorld	May-05	26	0.66	0.67
FLOSSMetrics	Sept-06	30	0.58	0.58
TEAM	Sept-06	30	2.95	4.16
EDOS	Oct-04	33	2.22	3.45
CALIBRE	June-04	28	1.50	1.65
SQO-OSS	Sept-06	24	1.64	2.47
PYPY	Dec-04	28	1.35	2.29
QUALOSS	Sept-06	30	2.05	2.95
		Total:	**25.13**	**37.28**

Table 1: Research Projects on Libre Software Funded under the Scope of the 6th Framework Programme (source: <http://cordis.europa.eu/ist/st/projects.htm>).

In summary, libre software has an important impact on the economy of Europe. This impact will grow in the following years. Furthermore, libre software offers a very good opportunity to gain more knowledge on the software development process, with the final goal of getting the scientific base for a true Software Engineering. Currently there exist many research projects that are trying to address and overcome these problems, with the financial aid of the European Commission through the 6th and 7th editions of the Framework Programme. We think that the European Commission should keep this aid in subsequent editions.

References

[1] Fred Brooks. "The Mythical Man-Month: Essays on Software Engineering". Addison-Wesley. 1995. ISBN 0-201-83595-9.

[2] Rishab A. Ghosh et al. "The impact of Free/Libre/Open Source Software on innovation and competitiveness of the European Union". European Commission, 2007. <http://flossimpact.eu>.

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

Andrea Capiluppi and Martin Michlmayr

This article was previously published in IFIP International Federation for Information Processing Volume 234 (2007), eds. J. Feller, B. Fitzgerald, W. Scacchi, A. Sillitti, pp. 31-44. It is reproduced with kind permission of Springer Science, Business Media and the authors.

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.

Keywords: Open Source, Software Developers, Software Evolution, Software Process, Stages.

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.

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

Authors

Andrea Capiluppi obtained his Ph.D. from the *Politecnico di Torino*, Italy. In October 2003 he was a visiting researcher in the *Grupo de Sistemas y Comunicaciones* of the *Universidad Rey Juan Carlos*, Madrid, Spain. From January 2004 to the present he has been a visiting researcher in the Department of Maths and Computing at the Open University, UK, working in collaboration with Drs. Juan Ramil, Neil Smith, Helen Sharp, Alvaro Faria, and Sarah Beecham. This appointment has been renewed until December 2008. In January 2006, he joined the University of Lincoln as a Senior Lecturer. <acapiluppi@lincoln.ac.uk>.

Martin Michlmayr has been involved in various free software projects for over 10 years. He used to be the Volunteer Coordinator for the GNUstep Project and acted as Publicity Director for Linux International. In 2000, Martin joined the Debian Project, and he was later elected Debian Project Leader (DPL) as which he acted for two years. Martin holds Master degrees in Philosophy, Psychology and Software Engineering, and earned a PhD from the University of Cambridge. He's working for HP as an Open Source Community Expert. <martin@michlmayr.org>.

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 an OSS project does not operate in the context of a community of volunteers. All the characteristics of cathedral style development (like requirements gathering, 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].

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].

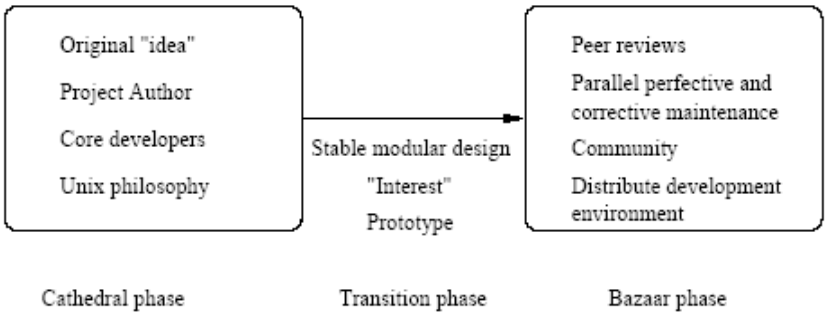


Figure 1: OSS Development Lifecycle.

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 output obtained from input provided, the second on what new developers tend to work on when joining an OSS project. The research questions can be formulated as follows (metrics used to assess each question are also provided):

1) 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.

2) Research question 2: new developers, when join-

ing 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 predefined 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 annotating 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, YYYYMMDD, as in "20001231". 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 plaintext 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 developer 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:

1) Input metrics: the effort of developers was evaluated by counting the number of unique (or distinct, in a SQLlike 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.

2) 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.

3) 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,

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

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 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.

Attribute/System	Arla	Wine
Earliest found entry	October 1997	July 1993
Latest studied entry	March 2006	March 2006
Change or creation points	7.000	88.000
Global, distinct developers	83	880

Table 1: Summary of Information in the two Systems.

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.

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.

3.2 Bazaar Phase

The aim of many OSS projects is to reach a stage

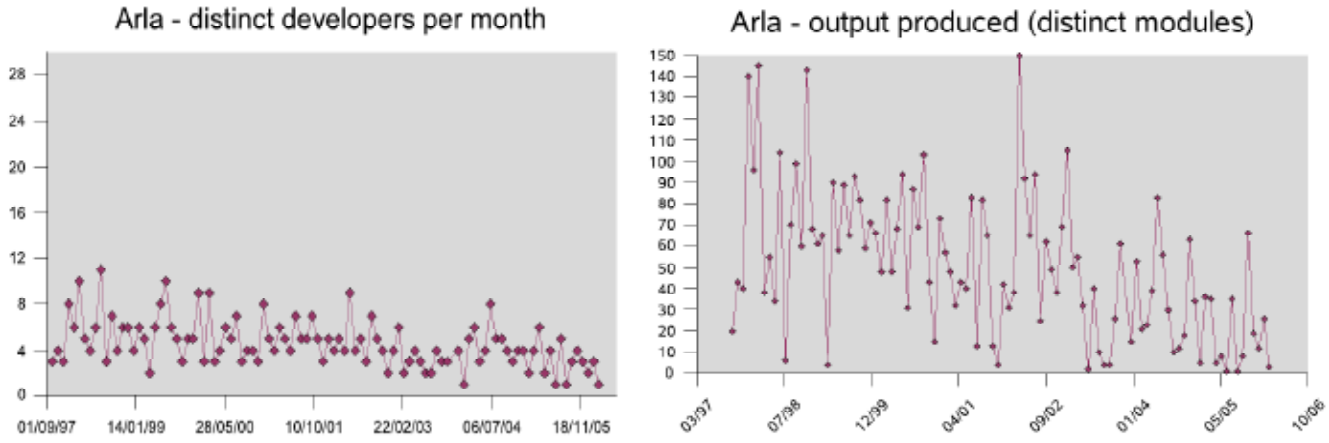


Figure 2: Development Input (left) and Output Produced (right) in Arla.

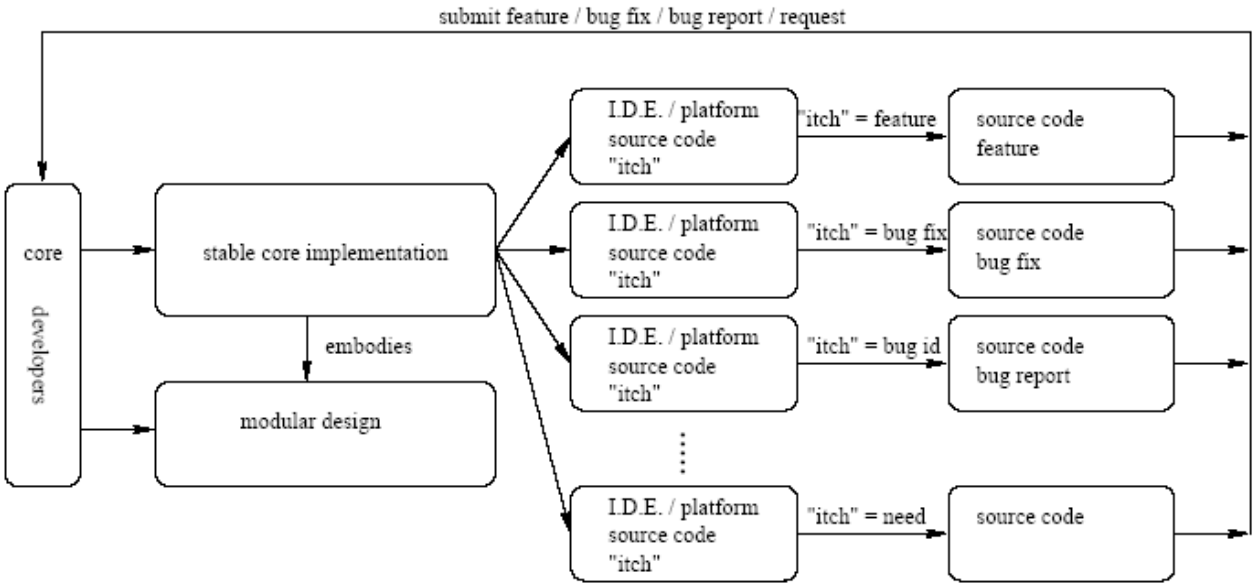


Figure 3: Detailed Bazaar Phase.

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 an 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, authors is observed. 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

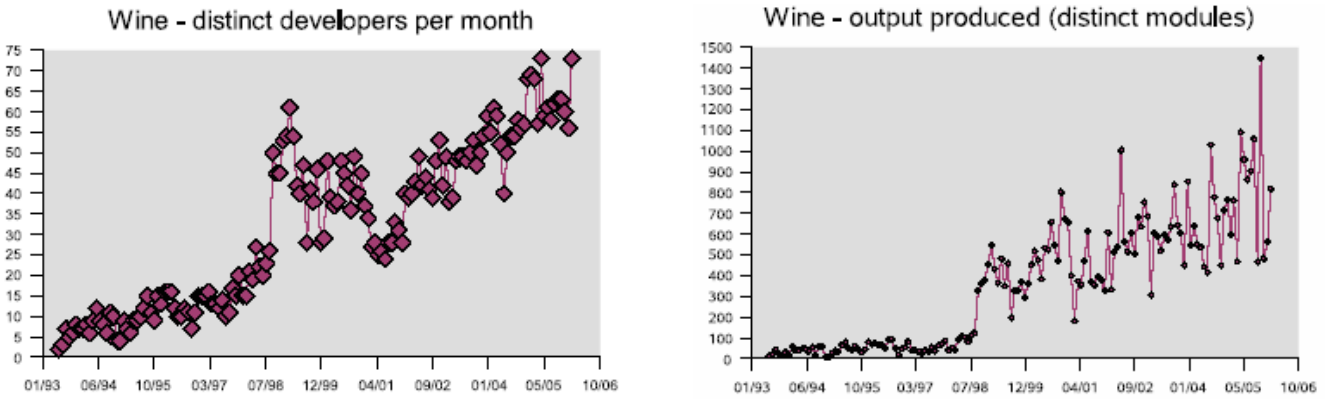


Figure 4: Development Input (left) and Output Produced (right) in Wine.

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 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 behaviour 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 4 (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 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.

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 5 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 6 is a description, on a percentile basis, of the modules as handled by newest developers.

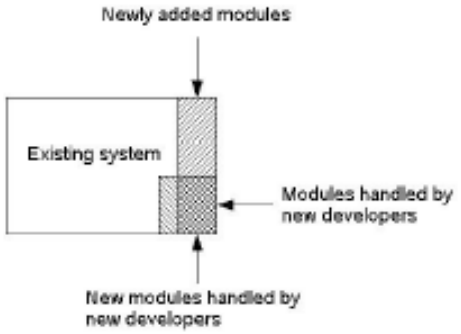


Figure 5: Design of Research Question 2.

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 box-plot of Figure 6). Also, the average value of the box-plot 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 7, right). A growing pattern is detected, similar to the one observed in the global evolution of the system (Figure 4): 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.

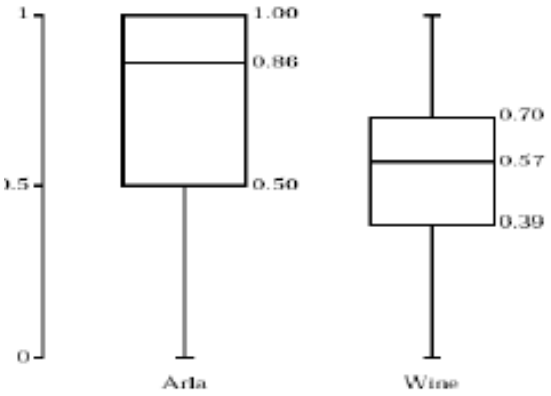


Figure 6: Description of Effort for new Developers.

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 box-plot of Figure 6). 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 7, 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.

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

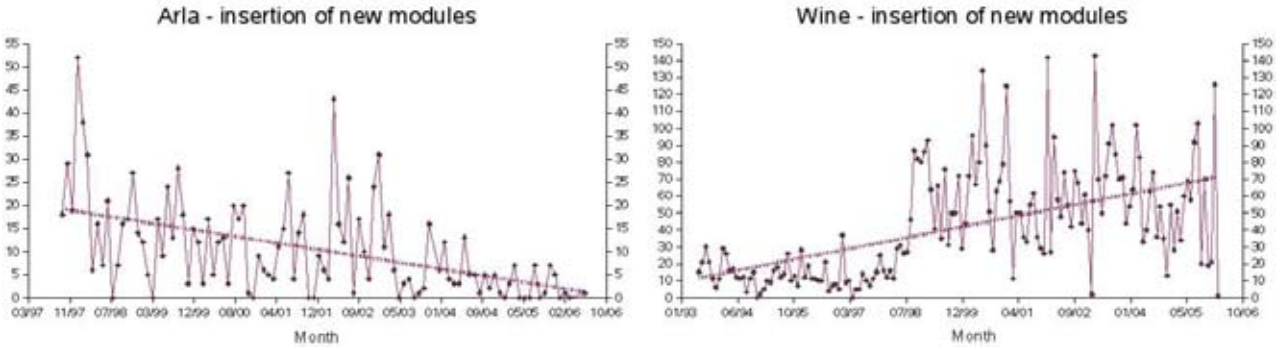


Figure 7: Creation of new Modules in the Arla and Wine Systems.

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, 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, 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, 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, M. M. Lehman. A model of large program development. IBM Systems Journal, 15(3):225-252, 1976.

[5] M. Bergquist, 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, pages 65-74, Amsterdam, Netherlands, 2003.

[7] A. Capiluppi, P. Lago, 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, 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, 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. ISBN: 1-57610-490-7.

[12] D. M. German. An empirical study of finegrained 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, 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, 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, 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, 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, L. A. Belady, editors. Program evolution: Processes of software change. Academic Press Professional, Inc., San Diego, CA, USA, 1985. ISBN: 0-12-442440-6.

[21] L. Lopez, J. G. Barahona, I. Herraiz, 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, 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 93102, 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. ISBN: 978-1-58603-559-4.

[25] M. Michlmayr, 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, 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, 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. ISBN: 1-56592-724-9.

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

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

[31] I. Stamelos, L. Angelis, A. Oikonomou, G. L. Bleris. Code quality analysis in opensource 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. ISBN: 1-56592-582-3.

The Commons as New Economy and what this Means for Research

Richard P. Gabriel

This article was previously published in the Proceedings of the First International Workshop on Emerging Trends in FLOSS Research and Development. FLOSS '07. ISBN: 0-7695-2961-5. Digital Object Identifier: 10.1109/FLOSS.2007.14. It is reproduced with kind permission of IEEE and the author.

Suppose the entire social and commercial fabric supporting the creation of software is changing—changing by becoming completely a commons and thereby dropping dramatically in cost. How would the world change and how would we recognize the changes? Software would not be continually recreated by different organizations, so the global "efficiency" of software production would increase dramatically; therefore it would be possible to create value without waste, experimentation and risk-taking would become affordable (and probably necessary because firms could not charge for their duplication of infrastructure), and the size and complexity of built systems would increase dramatically, perhaps beyond human comprehension. As important or more so, the activities of creating software would become the provenance of people, organizations, and disciplines who today are mostly considered consumers of software—there would, in a very real sense, be only a single software system in existence, continually growing; it would be an ecology husbanded along by economists, sociologists, governments, clubs, communities, and herds of disciplines. New business models would be developed, perhaps at an alarming rate. How should we design our research to observe and understand this change? There is some evidence the change is underway, as the result of the adoption of open source by companies who are not merely appreciative receivers of gifts from the evangelizers of open source, but who are clever thieves re-purposing the ideas and making up new ones of their own.

Keywords: Business Strategies, Commons, Intellectual Property, New Business Models, Open Source, Software License, Software Production, Source Code.

1 Introduction

Sometimes something new happens at a scale that both researchers and practitioners are either unable or unwilling to observe. An example of this in recent memory has been the emergence of emergence as a field of study, in the form of complexity science. For centuries a sort of phenomenon that is now regard as possibly central to many scientific disciplines was simply not observed or was considered not worthy of serious thought.

Researchers in and practitioners of open source[1] are enamored of licensing, tools and their usage, community building, and how effective and efficient the open-source methodology is at producing software. However, something much larger is going on that could be changing the landscape of computing and not just adding some knowledge to the discipline of software engineering.

Over the last 10 years, companies have been contributing a stupendous amount of software to (let's call it) the open-source world. For example, Sun Microsystems recently computed that, using conventional means for assigning a monetary value to source code, it has contributed over $1 billion in code. IBM and possibly other large corporations are not far behind. Of particular interest is that Sun has made a decision to open-source all of its software, and it appears they are well on their way to doing that. At the same time, Sun is not placing all of its revenue expectations on their hardware: they expect to make money with their software.

Author

Richard P. Gabriel received a PhD in Computer Science from Stanford University in 1981, and an MFA (Master of Fine Arts) in Poetry from Warren Wilson College in 1998. He is a Distinguished Engineer at IBM Research, looking into the architecture, design, and implementation of extraordinarily large, self-sustaining systems as well as development techniques for building them. Until recently he was President of the Hillside Group, a nonprofit that nurtures the software patterns community by holding conferences, publishing books, and awarding scholarships. He is on Hillside's Board of Directors. He is author of four books and a poetry chapbook. He has won several awards, including the AAAI/ACM Allen Newell Award. And he is the lead guitarist in a rock 'n' roll band and a poet. <rpg@{dreamsongs.com | us.ibm.com}>.

2 Sun: A Case Study (Brief Overview)

Sun started in 1982 as a company based on open

[1] I use this term for simplicity and to avoid politics.

standards and commodities: BSD Unix, Motorola 68000 processors, and TCP/IP. In the late 1990s it began to experiment with open-source ideas and true open source: Jini (not true open source, but an interesting experiment in open-source concepts and practices combined with strategies for creating markets), Netbeans, Juxta, and OpenOffice were early experiments, followed by Glassfish, Grid Engine, OpenSparc, OpenSolaris, Open Media Commons, and most recently Java.

Throw in Java.net and an interesting landscape emerges. Sun is clearly experimenting with the whole concept of the commons. OpenSparc is a hardware design that was licensed under an open-source license for the purpose of creating markets; Open Media Commons is primarily a DRM open-source project, but it is also looking at the question of what intellectual property rights means in the 21st century. Java.net is a sort of meta-community aimed at creating markets around Java. Solaris and Java are considered Sun's software crown jewels.

Throughout this experimental era at Sun (which is still going on) there were emphases on governance and business models.

Sun is pushing four open-source-related business strategies:

- To increase volume by engaging software developers and lowering the barriers to adoption.
- To share development with outside developers and established open-source projects for software required by Sun's software stacks.
- To address growing markets whose governments or proclivities demand open source, such as Brazil, parts of the European Union, Russia, India, and China
- To disrupt locked-in markets by providing open-source alternatives.

Sun makes an interesting set of observations about how the point has changed over time where monetization of software happens. In the 1970s, software was primarily part of a complete hardware package. People would buy a complete system (hardware and software). In many cases, hardware companies would provide the source code for their customers to customize—and nothing was considered unusual about this.

During the two decades from 1980 to 2000, hardware companies started to unbundle their software, and software companies sprang up to sell software to do all sorts of things, including operating systems. What these two periods had in common was that software was monetized at the point of acquisition. And it seemed at the time there was no choice: you wanted to use something, so you needed to buy it first.

With open source and the right business models, this can change, and that change started in the early 2000s. Open source is typically free to use—that is, no cost. However, there are auxiliary things companies, and in some cases individuals, willing or eager to pay for: support and maintenance, subscription for timely updates and bug fixes, indemnification from liability, and patent protection. In these cases, monetization can occur when the final product is deployed. That is, in such cases it costs nothing to explore an idea for a product to the point of putting it completely together for sale or distribution. Then, if the producer wishes, one or several of these services can be purchased.

By delaying some of the costs of coming up with new products and possibly new companies, likely many more new ideas can be explored and considered over the entire market. The barriers for experimentation are very low.

The full repertoire of business models Sun has identified are as follows:

- Subscription (as described above) including indemnification and patent protection by extending a company's umbrella of intellectual property over parties who subscribe.
- Dual license, in which newer versions of the code are sold and older ones are open source.
- Stewardship, in which a standard is used to attract developers using the standard and to whom other products and services are sold.
- Embedded, in which the code is part of something else (usually hardware) that is sold.
- Consulting, in which a person's or company's expertise, in particular source code base, is sold as, typically, heads-down programming services.
- Hosting, in which services provided by open-source software is running on servers and access to the running services are sold or other revenue streams are attached to the running code (like advertisements).
- Training and education—of the source base and also of open-source methodologies.

Sun open-source theoreticians view these observations as implying a virtuous cycle in which by finding a place for added value in code in the commons, a company (or person) can create a monetization point without having to invest alone in a large code base, and thereby produce a product or service at lower overall cost.

3 What This Means for Software

Suppose that Sun is not an isolated situation and that companies and other organizations (including individuals) are preparing to alter their business and software development models to be based on the Sun-described virtuous cycle. How would the entire enterprise of producing software change and what would this mean for software engineering?

Let's paint the picture. The vast majority of software would be in the commons and available for use. Nothing much would be proprietary. There would be pressure from the customer base for there to be some unifications

or simplifications. For example, why would there need to be multiple operating systems aside from the needs of different scales, real-time, and distributed systems (for example)? On the other side, finding new value might cause pressure on firms to fork source bases to create platforms or jumping off points for entire categories of new sources of value. How would this balance play out?

Because the barriers to entry to almost any endeavour would be so low, there will be many more players (including small firms, individuals) able to be factors in any business area. With more players there would be more opportunities for new ideas and innovations. How will these play out in the market? Will, perhaps, firms try to become repositories of intellectual property in order to offer the best indemnification? Will other entities like private universities or pure research labs become significant players because they can offer a potent portfolio of patents to use to protect their clients? Looking at large portfolios such as owned by IBM or Microsoft, it would seem that they would continue to dominate; however, in new or niche areas, small organizations or even individuals could hold the key patents.

Some obvious considerations immediately come up. What about licensing? At present large systems are put together from subsystems (to pick a term) licensed under different licenses. What is not permitted is to be able to mix pieces from differently licensed source bases. Will there be pressure to put all code under the same license or will the pressure be the other way—to create new licenses for specialized purposes?

4 What This Means for Software Engineering

Because few companies would "own" an entire system or application area, there could be some pressure on code bases to drift regarding APIs, protocols, data formats, etc. And if so, where would the countermanding pressure come from? Would standards bodies handle it, would governance structures like the Apache Foundation or the IETF be created? Or would firms spring up to define application or system structure as was done with the personal computer in the early 1980s. In that case, a set of design rules were set up by IBM stating what the components of a PC were and how they interacted [1]. This enabled markets to form around the different components and the nature of design in computer systems changed. Today this way of looking at design has spawned a new approach to software engineering problems: economics-driven software engineering.

Software and computing education would change because all the source code would be available for study (and even improvement as part of the teaching/learning process).

In this way, developers would be better educated than they have ever been before.

Programming would become less a matter of cleverness and invention, and more a process of finding existing source code that's close and either adapting or adapting to it. Licensing would either help or hinder this.

With pressure lessened to build everything from scratch, it would be possible to construct larger and larger systems with achievable team sizes. This would bring out the issues and challenges associated with ultra-large-scale systems[2]. To quote from the call for position papers for a workshop on this topic [2][3]:

In a nutshell, radical increases in scale and complexity will demand new technologies for and approaches to all aspects of system conception, definition, development, deployment, use, maintenance, evolution, and regulation. If the software systems that we focus on today are likened to buildings or individual infrastructure systems, then ULS systems are more akin to cities or networks of cities. Like cities, they will have complex individual nodes (akin to buildings and infrastructure systems), so we must continue to improve traditional technologies and methods; but they will also exhibit organization and require technology and approaches fundamentally different than those that are appropriate at the node level. The software elements of ULS systems present especially daunting challenges. Developing the required technologies and approaches in turn will require basic and applied research significantly different that that which we have pursued in the past. Enabling the development of ULS systems—and their software elements, in particular—will require new ideas drawing on many disciplines, including computer science and software engineering but also such disciplines as economics, city planning, and anthropology.

The switch from proprietary to commons-based software would hasten the age of ultra-large-scale systems which will differ qualitatively because of their massive scale. If that happens, the inadequacies of our tools including programming methodologies and languages would be placed in high relief.

5 What This Means for Research

The habit of research in computing is to look deeply and narrowly at questions. In a sense, researchers love puzzles. Gregory Treverton wrote this about puzzles versus mysteries in a paper on/for the intelligence community [4]:

Now, intelligence is in the information business, not just the secrets business, a sea-change for the profession. In the circumstances of the information age, it is time for the intelligence community to "split the franchise" between puzzles and mysteries. Puzzles have particular solutions, if only we had access to the necessary (secret) information. Puzzles were the intelligence community's stock-in-trade during the Cold War: how many missiles does the Soviet Union have? How accurate are they? What is Iraq's order of battle? The opposites of puzzles are "mysteries", questions that have no definitive answer even in principle. Will North Korea strike a new nuclear

[2] This is the topic of a workshop I'm leading on Tuesday at ICSE.

bargain? Will China's Communist Party cede domestic primacy? When and where will Al Qaida next attack? No one knows the answers to these questions. The mystery can only be illuminated; it cannot be "solved."

Finding evidence of the sea-change from proprietary software to commons-based software in the commercial world is part of a mystery, not a puzzle, and so our traditional methods might not hold up well. But certainly studying the engineering methods open-source projects use will not illuminate the larger context—that context being how the entire enterprise of creating software changes when corporations change their business models to embrace the commons. The concerns of firms are not the same as the concerns of someone using a bug-tracking tool, editing code with Emacs, and automating a tricky part of the testing process. Moreover, because bottom-line concerns dominate sticking to certain ideals of engineering, for example, we are likely to see ideas we in the software engineering community have not thought of.

Here is a small example, again from the Sun case study. A Japanese automobile manufacturer contacted Sun's Open Source Group to learn about open-source. The group was responsible for the creation of the bulk of the company's applications. They claimed to not have a single coder in their direct employ, but outsourced—primarily to India. They were concerned that the Indian companies they were using were not as adept with interpreting the specs they were given as made financial sense for the car company. So the VP of the group was interested whether the Sun Open Source Group could help them figure out how to impose an open-source methodology (but not reality) on the Indian outsourcing companies so that the applications group could monitor progress, run the nightly builds, observe email and wiki-based communications, and etc, to both judge how the project was going and to correct it on the fly, perhaps by using open-source techniques.

Not a line of code would be released to the outside world; there would be no license. It would be simply a management tool. Researchers who would notice and report on such innovations and activities would come from a business school, or would be economists or perhaps anthropologists. Therefore what I see required is a broader view, a more interdisciplinary view—this is in concert with the conclusions reached by the authors of the ultra-large-scale systems report.

Another part of the sea change is that software researchers would be able to do real science on naturally occurring software, systems, frameworks, etc. For example, it would start to make sense to get a handle on how many times a piece of data is transcoded on its way from a database to a client screen somewhere, a number that could be very high particularly if the system doing the overall transmission were made of a number of separately developed frameworks. Today, gathering such information requires a special relationship with a corporation—a relationship that I suspect is quite rare.

6 Conclusions

One can wonder whether Sun's directions are predictive or iconoclastic. If the latter, then Sun is merely a curiosity; but if the former, it behooves those of us who straddle the research / practitioner boundary to figure out a sort of research program that will help us notice the changes in order to record and study them.

References

[1] C. Baldwin, K. Clark. Design Rules: The Power of Modularity. MIT Press, 1999. ISBN: 0262024667.

[2] First ICSE Workshop on Software Technologies for Ultra-Large-Scale (ULS) Systems. <http://www.cs.virginia.edu/~sullivan/ULS1/>.

[3] The Software Engineering Institute (SEI). The Software Challenge of the Future: Ultra-Large-Scale Systems, June 2006. <http://www.sei.cmu.edu/uls/>.

[4] Gregory F. Treverto. Reshaping Intelligence to Share with "Ourselves". Commentary No. 82, Canadian Security Intelligence Service, July 2003. <http://www.csis-scrs.gc.ca/en/publications/commentary/com82.asp>.

Libre Software for Research

Israel Herraiz-Tabernero, Juan-José Amor-Iglesias, and Álvaro del Castillo-San Félix

Traditionally, research projects tend to be less than transparent, only showing to the public selected deliverables but no internal information. Normally no information about how the research project is progressing is available as public data. Even the partners of the project tend to be unaware of how the other partners are getting on. In this respect, research projects are similar to traditional software development projects. Research projects in the field of Information Society Technologies share some features with libre (free / open source) software projects, such as global distributed development and the possibility of teleworking. In the light of the above, in this paper we present a proposal to manage research projects, adopting methods used in the libre software community, and using libre software tools. Our methodology facilitates communication flows between the various partners of the project, even if they are geographically dispersed, and also allows selected internal information to be shared with the general public. Furthermore, by adopting this methodology, several additional possibilities arise, among which are automated public activity reports, project progress analyses, and technological watching and foresight techniques. We firmly believe that this new approach to managing research projects presents a number of advantages over traditional organization methods, and may improve the performance of research projects.

Keywords: Framework Programme, Free Software, Libre Software, Open Source, Research, Research Management.

1 Introduction

Within the scope of the 6th Framework Programme (6FP), libre (free / open) source has begun to arouse interest, and several projects have been studying the phenomenon with a view to increasing knowledge and improving software development. Many of the good practices applied in libre software projects could be adapted to the management of complex environments. We think that one of these complex environments could be research projects themselves.

In a research project, people from different countries work in coordination to achieve the goals of the project. These people, often in different geographical locations, need to work on the same documents or on the same pieces of software, and consequently need to be aware of the work of the other partners to ensure an efficient division of work.

Traditionally, however, research projects tend to be less than transparent. Partners are not fully aware of what

Authors

Israel Herraiz-Tabernero is a Ph.D. student at the *Universidad Rey Juan Carlos,* Madrid, Spain. His research is related to the evolution of libre software projects. In particular, he is using time series analysis and other statistical methods to characterize the evolution of software projects. He has participated in several research projects funded by the Framework Programme of the European Commission (QUALOSS, FLOSSMetrics, QUALIPSO, CALIBRE). He has also collaborated on other projects funded by companies such as Vodafone and Telefonica. He has participated in the writing of manuals about managing and starting libre software projects. For example, together with Juan José Amor and Gregorio Robles he wrote a manual for the *Universitat Oberta de Catalunya's* Master Programme in Free Software. He has been a reviewer for the IEEE Africon 2007 among other conferences and for the journal IEEE Transactions on Software Engineering. He is currently a research and teaching assistant at the *Universidad Rey Juan Carlos*, pursuing his PhD on the evolution of libre software. He also coordinates the programme of the Libre Software Master offered by the *Universidad Rey Juan Carlos*, in collaboration with Igalia and Caixa Nova. <herraiz@gsyc.escert.urjc.es>.

Juan-José Amor-Iglesias has an M.Sc. in Computer Science from the *Universidad Politécnica de Madrid* and he is currently pursuing a Ph.D. at the *Universidad Rey Juan Carlos*, where he is also a project manager. His research interests are related to free software engineering, mainly effort and schedule estimates in free software projects. Since 1995 he has collaborated in several free software related organizations: he is a co-founder of LuCAS, the best known free documentation portal in Spanish, and Hispalinux, and he also collaborates with Barrapunto.com and Linux+. <jjamor@gsyc.escert.urjc.es>.

Alvaro del Castillo-San Félix has been in Free Software communities for over 12 years. His first experience was with the Linux kernel and all his professional life has been related to Free Software projects. He has worked on the foundation of projects such as Barrapunto.com and large communities such as GNOME Hispano and Mono Hispano. He worked as project manager for the first version of LinEx and during his role as Architectural Director in LambdaUX software company he participated in the Compatiblelinux.org project. He works in the GNOME community, mainly on the Planner project where he has spent over four years coding, and also on projects like FSpot and Evolution. He is currently coordinating the European Free Software Projects in the GSyC/LibreSoft research group at the URJC university where he lectured for four years in distributed systems. <acs@gsyc.escert.urjc.es>.

the rest of the partners are doing, the general public may access only selected documents, commonly referred to as deliverables, and not all deliverables are made available to the public.

This is a serious problem. First of all, at least within the scope of the 6th Framework Programme, research projects are publicly funded. Therefore all results (not only the final deliverables but all the work done in the project) should be available to those who are paying for the project.

Furthermore, at least in the case of libre software projects in the 6th Framework Programme (and probably in other fields too), several projects partially share the same goals and need access to the same sources of information. These projects could gain from other similar projects if they could access the internal documents and information generated by each project. Think of the analogy with the libre software world: if developers know that they can reuse a piece of source code available in any other project, they can simply take it and adapt it for their own purposes.

Because of all these issues we propose a methodology to adapt the practices applied in the libre software community to the management of research projects. Our methodology is intended to be adopted by all of the partners of a given project. The paper continues as follows. The next section describes the characteristics of a typical research project. Section 3 describes the needs of a research project and proposes tools to meet these needs. Section 4 explains how to organize the work and the environment of tools supporting that work, based on the experience of our research work. Finally, Section 5 draws some conclusions.

2 Structure of a Research Project

In this section we describe the structure of a typical research project. We take as examples our experience in research projects within the scope of the 6th Framework Programme.

Research projects are proposed and developed by a number of partners from different countries. This gives rise to the first problem we encounter when working on a project: language. English tends to be the language chosen for all communication between partners and for all internal and public documents generated.

The work is divided into *workpackages*. Each partner may lead one or more workpackages and all partners will participate in at least one workpackage. These workpackages will contain both *milestones* and *deliverables*. Milestones are key dates on which a certain piece of work is due. Deliverables are documents (although they may also be software, a database, etc) forming part of the final outcome of the project. Some deliverables are public, some internal to be used by the project partners, and others are intended to be delivered to the sponsor of the project (in 6FP's case, the European Commission).

The work required to produce the deliverables usually needs to be performed by various partners in coordination. Usually, one of the partners acts as coordinator and looks after all the economic aspects of the projects, while ensuring that all the work to be performed by each partner is completed according to the workplan and in a timely fashion.

The key to a research project is coordination: the various partners need to coordinate with the rest of the partners and it is very important for all partners to be aware of the work performed by the others. Of course, each partner is responsible for its own work and for delivering it on time.

3 Needs of a Research Project

Certain tools are required if the project is to be developed as described above. Firstly we will talk about the general concepts behind what a research project needs, before going on to propose a number of libre software tools to meet those needs.

- **Website**

First of all, the dissemination requirements of a publicly funded project should be covered by a website. It is usual to build a content management system (CMS) to make it easier for the partners to publish documents and for the general public to access them.

From the Wikipedia page on CMS[1]:

A content management system (CMS) is a system used to manage the content of a Web site. CMSs are deployed primarily for interactive use by a potentially large number of contributors. For example, the software for the website Wikipedia is based on a wiki, which is a particular type of content management system.

The website should also be capable of distinguishing between public and private documents, making private documents available only to selected users (typically the partners of the project).

- **Mailing list**

Secondly, in order to facilitate communication between partners, a mailing list is required. Sometimes it is a good idea to set up two different mailing lists, one for all the people involved in the project and another limited to the core group members. In our opinion, there are some strategic decisions regarding the research project that should only be discussed by the core group and not by all the researchers taking part in the project.

If the group of people working together is greater than 4 or 5, it is essential to have a mailing list. Mailing lists also provide other advantages such as a record of past messages that can be useful when new members join the group to work on the project after it has started. Usually there will be two mailing lists, one for everyone involved in the project and another just for the core group. If the research group is small, it may be enough to have just one mailing list.

- **Version control system**

There is also a need for a repository of working docu-

ments and software (files of any kind in general) with version control capabilities. This makes it possible to recover past versions of the documents and to work on the same documents in coordination with other people. It is also a central point where anybody can find any document or file belonging to the project. This repository is not intended for the publication of deliverables but rather to help researchers work on documents in a coordinated fashion. Control version capabilities are crucial because different people work on the same document and it may be necessary to recover a past version of a document.

- **Wiki**

Another interesting tool is the use of *wikis*, which make it possible to work on documents using a web browser. From the Wikipedia page on wikis [2]:

A wiki is software that allows users to create, edit, and link web pages easily. Wikis are often used to create collaborative websites and to power community websites.

Wikis allow researchers to work on documents on the web using only a web browser. It is intended for lightweight documents. In our opinion, it is not an appropriate tool for writing deliverables but it is more than adequate for organizing the research group's knowledge base.

- **Issue tracking system**

Finally, an *issue tracking system* may also be useful. From the Wikipedia page on this subject [3]:

An issue tracking system [..] is a computer software package that manages and maintains lists of issues, as needed by an organization. Issue tracking systems are commonly used in an organization's customer support call center to create, update, and resolve reported customer issues, or even issues reported by that organization's others employees. An issue tracking system often also contains a knowledge base containing information on each customer, resolutions to common problems, and other such data.

In the case of a research project, the tracking system can be used by managers to assign tasks to people and other resources, and to monitor the progress of the work. This makes the life of the project manager easier and ensures that everybody is aware of the work performed by the rest of people in the group.

In our opinion, this is the basic set of tools that any group working on a research project should make use of. They make it easier to organize and monitor the group's work on a day-to-day basis. 3.1 Tools to Meet these Needs

- **Website**

For the first requirement (a website with CMS capabilities) there are a number of platforms available in the libre software community. A comprehensive list of libre software alternatives may be found at [5]. Most of them include the capabilities required by a research group, such as document repository with different profiles (public, private, and so on).

However, our recommendation is not included in the above mentioned list. We recommend using Plone [9].

- **Mailing lists**

With regard to mailing lists, we recommend using *GNU Mailman* which is a package for managing electronic mailing lists. It has a web interface to administrate the system and enables messages to be archived and accessed via a web interface. More information about GNU Mailman can be found on the relevant Wikipedia page (see [4]).

- **Version control repository**

For the version control repository we recommend *Subversion* [6] (also known as SVN). The main reason behind our choice is that Subversion integrates better with other tools and can be accessed using standard Webdav clients, supported by the file browsers of almost every operating system, although it is better to use a Subversion specific client so as to be able to make full use of its capabilities.

- **Wiki**

For wikis, in our opinion the solution of choice is the popular MediaWiki, the system used by Wikipedia itself [7] among others.

- **Issue tracking system**

Finally, for our issue tracking system, we recommend Trac [8]. What is even more interesting about Trac is that it can integrate a wiki, a subversion repository, an issue tracker, and a timeline for project planning. For instance, when submitting a ticket, it can be associated with a milestone in the project planning, with a given revision of a document in the SVN repository, or with the people involved in that ticket. The information is available in other web accessible formats: text format and RSS. In particular, RSS allows information to be processed automatically, which is useful for technological tracking and activity reporting systems.

There are however a great many alternatives for issue tracking systems.[10] includes a comprehensive list of tracking systems, broken down into various categories.

4 Organization of the Work

In this section we present how we used the tools mentioned in the previous section to meet our needs when

participating in some European projects.

First of all, this is the list of tools that we chose:

- Zope for our website.
- Mailman for the mailing lists.
- Subversion for the control version system.
- Trac for the wiki and the issue tracker. The SVN repository is integrated with Trac.

For the website, we developed our own solution, using Zope as a framework. The website does not meet the above mentioned requirements (document repository, profiles for different kind of users, etc). However, within the scope of the project, other solutions meeting these requirements were adopted. For instance, in some projects, Plone (which is based on Zope) was chosen.

In the case of mailing lists, we have three different mailing lists for each project:

- A list to which everybody working on the project is subscribed.
- A list containing only the core group managing the project.
- A list to which all partners are subscribed. This is useful when the trac only covers the work of one team but the project has several teams from different institutions working on it.
- A list of commit watchers. Every time a new commit is added to the version control system, a message with a summary of the commit is sent to this list. This allows everyone to be aware of the changes made to the repository.

For the mailing lists we use Mailman. The lists are usually configured as moderated for unsubscribed people to avoid junk emails. Some lists, such as core or partners lists, could be also configured as private (nobody can subscribe to the list or read the archives without authorization).

For the wiki and the issue tracking system, we use Trac. We also integrated the Subversion repository in Trac. We use the wiki for the project's knowledge base, and the tracking system to control, assign, and monitor the work in the project. Also, any electronic mails generated by this tool (when issue tickets are created or closed) are sent to the list used by the working team.

When managing several projects at a time, each one with its own trac, it is very useful to integrate the activity tracking of each project in a "planet" (an RSS aggregator[1]). Planets are very useful for seeing the recent activity of all projects in a single web page, by importing all RSS files representing the timeline content of each trac site.

Our team has modified Planet in order to integrate activity indicators as well. An activity indicator is a smiley which represents the most recent activity of a project. For

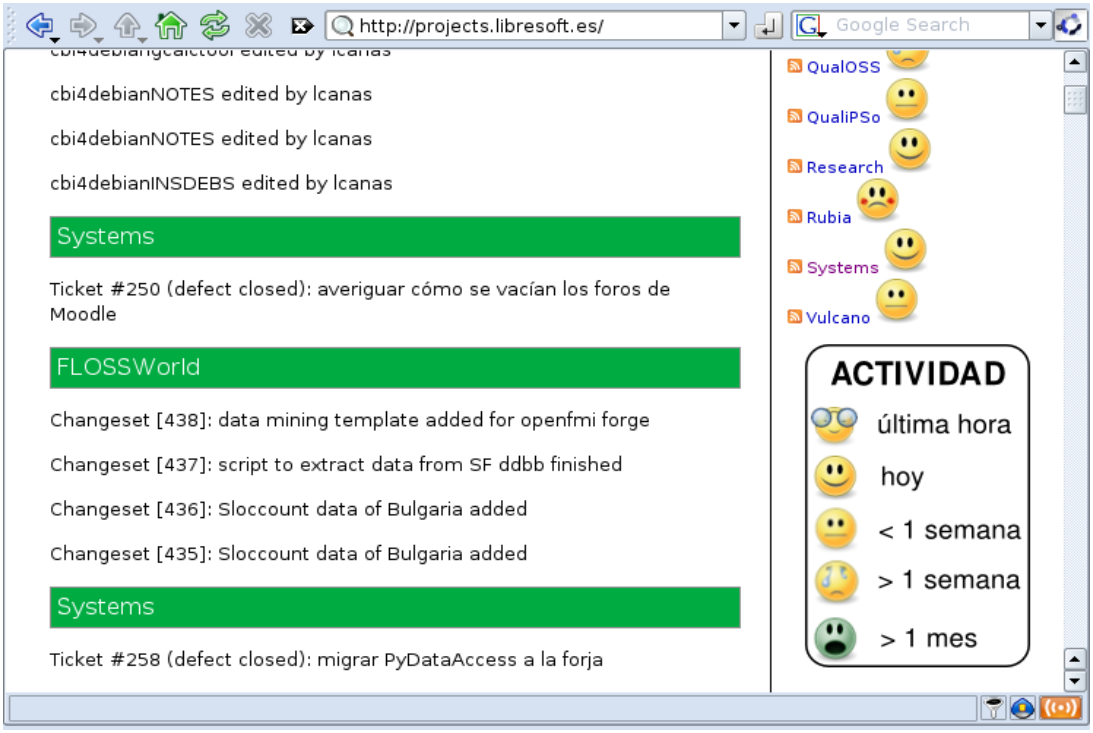


Figure 1: Planet Website (showing recent activity in the various projects and indicators related to this activity).

[1] The most commonly used, written in Python, is available at <http://www.planetplanet.org/>.

example, if a project has registered activity in the last hour, the smiley is laughing. But when the last activity is a day old, the face is more serious. There are several smileys until the worst case, representing a project which has not registered any activity in last month.

An example from this website is shown in figure 1. The left hand side displays a list of recent events, classified by project. On the right hand side, we can see a list of all the projects, with the indicator of the most recent activity. Below the list of projects there is a legend explaining the activity indicators.

The RSS feeds from the Trac tool of each project are integrated into a single website. This feed contains an entry for each event occurring in the Trac. It may be a ticket event (created, changed, etc), an event in the wiki (modification, addition or removal of a page), or an event in the Subversion repository (again modification, addition or removal).

This website has proved to be very useful for the group. First of all because it enables anyone working in the group to be aware of any recent work done in all the projects and who did it. Secondly, because the activity indicators act as a "motivator" for the various subgroups working on each project. For instance, if one group takes the lead in recent activity as shown by the indicators, another group may be encouraged to work harder to get back on top.

To sum up, we have implemented all the above mentioned tools and have realized their full potential. For example, our Trac websites integrate wiki, Subversion repository, and an issue tracking system. We also have a mailing list which receives a message every time a change occurs in any of the repositories. As we work on various projects we can consolidate the information about the recent activity of these projects in a single website. This means that everyone can be aware of the recent work performed by the rest of the group, regardless of which project they are working on. Furthermore, activity indicators act as motivators to maintain a high level of activity compared to other projects within our own research group.

However, we have to admit that due to external requirements we have not yet been able to fully open up our tools to the rest of the world. So we are not yet benefitting from sharing our knowledge with other partners working on different projects, although we are working towards that goal.

5 Conclusions

In this section we present a methodology and a set of tools to organize a research project and the various groups working on the project. Our methodology is based on the methods and tools used to manage and organize libre software communities.

Research projects should be as open as libre software projects are for two reasons: they are usually publicly funded and so they should be publicly available to everyone, and some projects may benefit from collaborating with other research projects, thereby making a more efficient use of public funding.

Our proposed methodology allows all information to be made publicly available. Not only the final deliverables but all the work done during the lifetime of the project.

The proposed tools make it possible to keep track of all the work performed during the entire lifetime of the project. These repositories of information on the research project open up new avenues to improve the efficiency of research projects; for example, the automated technological watching of research projects based on the trails available in the repositories of the project (website, mailing list, version control system, issue tracking, etc).

The proposed tools and methods also allow information to users and to the public to be filtered on the basis of different information access profiles.

Another strong point of this methodology is that it makes it possible to work remotely, as all the information is managed using the proposed tools and all the tools can be accessed remotely. Thus it would be possible for people visiting other partners or universities to continue working. It also enables various partners to work in coordination in spite of being in different countries.

The only drawback of our proposal is that it is only valid for Information and Communication Technologies. For instance, chemical or biological projects require people to work together at the same location. However, the tools may still be useful to organize some parts of the work, for example the management of deliverables.

In future work we will use the trails of the repositories of the projects on which we are working to build a technological watching system to track the research carried out on libre software. We are also planning to build tools to automate activity and participation reports based on the information provided by the repositories. In the near future we are also considering completely opening up our repositories to make the information available to anyone. At the moment, as we are working with other partners, that decision is not in our hands. In any event, all the results of our projects are offered under non-restrictive licenses, both for software and documents.

References

[1] <http://en.wikipedia.org/wiki/Content_management_system>.

[2] <http://en.wikipedia.org/wiki/Wiki>.

[3] <http://en.wikipedia.org/wiki/Issue_tracking_system>.

[4] <http://en.wikipedia.org/wiki/GNU_Mailman>.

[5] <http://www.opensourcecms.com/>.

[6] <http://en.wikipedia.org/wiki/Subversion_%28software%29>.

[7] <http://en.wikipedia.org/wiki/Mediawiki >.

[8] <http://en.wikipedia.org/wiki/Trac >.

[9] <http://plone.org >.

[10] <http://en.wikipedia.org/wiki/Comparisonofissuetrackingsystems >.

Technological Innovation in Mobile Communications Developed with Free Software: Campus Ubicuo

Javier Carmona-Murillo, José-Luis González-Sánchez, and Manuel Castro-Ruiz

Nowadays, wireless communications networks are one of the fastest growing segments of the communications field. The increasing demand for services and the need for mobility have changed the traditional model of Internet connectivity based only on access through fixed networks. Starting from both the portable devices and the current wireless access network position, we propose a system designed to provide mobility and ubiquity in a university campus environment, easily adaptable to all kind of organizations. In this paper we present Campus Ubicuo, a research, development and innovation project in mobile communications field. The project, which is developed using free software, aims to offer the user ubiquity through advanced communications services over wireless networks. Moreover, the project development has allowed researching into IP mobility and interference analysis produced by several wireless communications technologies.

Keywords: Free Software, IP Mobile, Mobility, PDA, Ubiquity, 3G.

1 Introduction

In recent years the research community has started to focus attention on free software as a basis for the development of its proposals and contributions. In the computing and communications field, this attention is especially useful if we take into account the characteristic development model used in free software [1].

On the other hand, the current demand of ubiquitous connectivity, no matter what the place, time or access technology, has made mobile communications necessary. There are three main features in this environment that must be taken in account: User mobility in wireless networks; the Quality of Service (QoS) of these communications; and finally, security issues for the information transmitted across mobile networks. The Campus Ubicuo[1] project proposes to connect them by means of free software, which contributes some new advantages.

This project is the outcome of the experience obtained over some years of research into communications, mobility and free software, and comes to take advantage of the mobility and portability possibilities of PDA (Personal Digital Assistant) devices, mobile phones and laptops. Apart from these devices, also are included technologies like GSM (Global System for Mobile Communications),

Authors

Javier Carmona-Murillo is a PhD student in the Computing Systems and Telematics Engineering Department at the University of Extremadura, Spain, where he received his engineering degree in Computer Science (2005). His main research topics are broadband networks, QoS provision in mobile networks and IP mobility. He is currently working on the Campus Ubicuo project and he is also doing research support work in the Advanced and Applied Communications Engineering Research Group (GITACA) <http://gitaca.unex.es>. <jcarmur@unex.es>.

José-Luis González-Sánchez is a full time Associate Professor of the Computing Systems and Telematics Engineering department at the University of Extremadura, Spain. He received his Engineering degree in Computer Science and his Ph.D degree in Computer Science (2001) at the Polytechnic University of Cataluña, Barcelona, Spain. He has worked for some years at several private enterprises and public organizations, as a System and Network Manager. He is the main researcher of the Advanced and Applied Communications Engineering Research Group (GÍTACA) of the University of Extremadura. He has published many articles, books and research projects related to computing and networking. Currently he is the president of CPIIEx (Professional association of the engineering in computer science of Extremadura). <jlgs@unex.es>.

Manuel Castro-Ruiz received the Technical Engineering degree in Computer Science from the University of Extremadura, Spain. He completed a Masters in Strategic Administration and Innovation Management at the Autonomous University of Barcelona. In the last 10 years he has worked in several consultancy companies in the Information and Communications Technology sector in Madrid, Andalucía and Extremadura. He has managed Information Systems projects for private enterprises and public organizations. Currently, he is the Sadiel S.A representative in Extremadura. <mcruiz@sadiel.es>.

[1] Campus Ubicuo <http://gitaca.unex.es/cubicuo> is a project developed in agreement between the Advanced and Applied Communications Engineering Research Group (GITACA) of the University of Extremadura <http://gitaca.unex.es> and the company SADIEL S.A. This work is sponsored in part by the Regional Government of Extremadura (Education, Science and Technology Council) under GRANT Nº PDT05A041.

GPRS (General Packet Radio Service), UMTS (Universal Mobile Telecommunications System), Bluetooth or WiFi (Wireless Fidelity).

This paper presents the work developed in the project, as well as the research tasks carried out, organized as follows: In Section 2 we describe each Campus Ubicuo subproject, whereas research task related to Campus Ubicuo

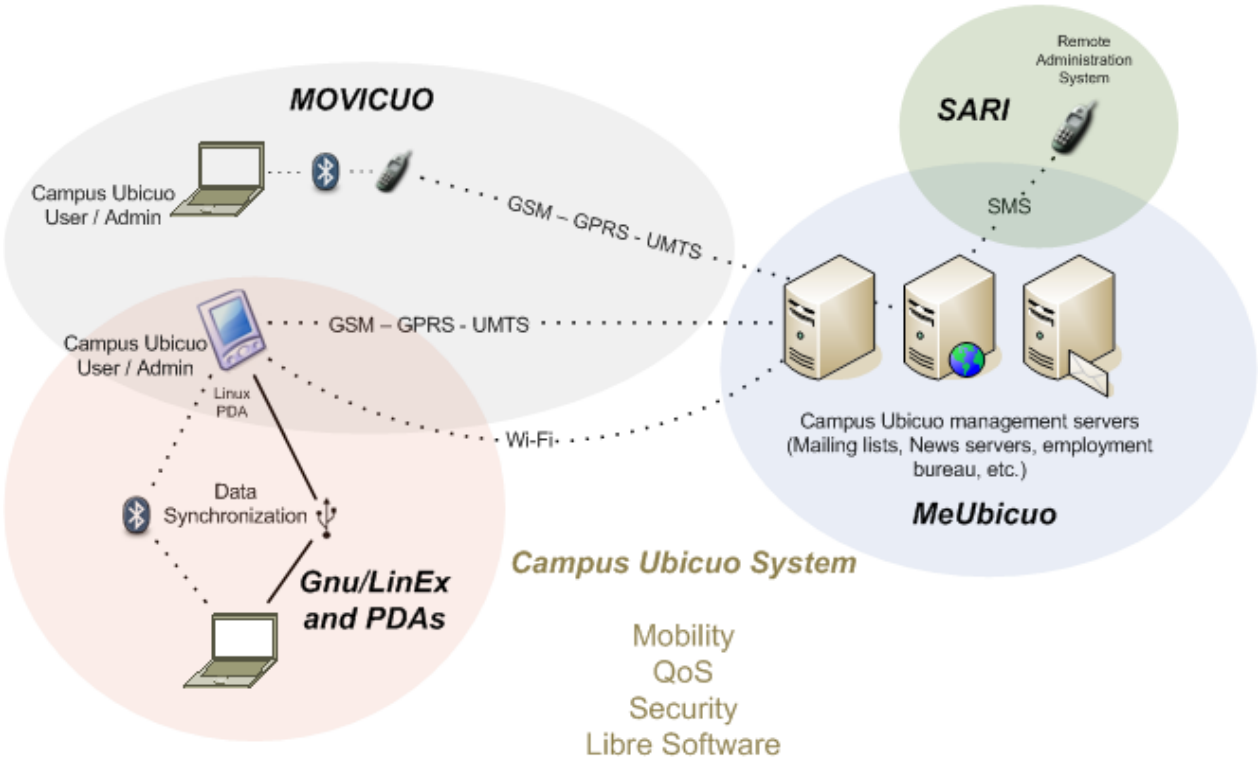


Figure 1: Campus Ubicuo Architecture.

are detailed in Section 3. Finally, conclusions and future work are presented in Section 4.

2 Campus Ubicuo Development

2.1 System Architecture

Figure 1 shows the global architecture of the proposed system. This is an infrastructure over which ubiquity services can be offered through mobile communications technologies. In the figure we can see the four pillars upon which Campus Ubicuo rests: mobility, QoS, security and free software.

In following sections, we describe each task in which the project development has been split: Movicuo, GNU/LinEx in PDA, SARI and MeUbicuo.

2.1.1 Movicuo

In Campus Ubicuo, we devote a great amount of our efforts to develop software that contributes mobility to users of free operating systems like GNU/Linux. Movicuo is the project subsystem where we carry out these tasks.

Nowadays, the technologies that have mobility as an inherent feature are mobile phone networks. GPRS (2.5G) complements the design of GSM (2G) by adding a packet switched network to carry data traffic; moreover it allows QoS parameter negotiation. However traffic rates achieved by GPRS (171.2 Kbps under ideal conditions) [2] are not adequate to support some particular services, as for example multimedia traffic. UMTS (3G) uses a new media access method that adds complexity but allows achievement of higher traffics rates [3]. In order to establish a 2.5G or 3G connection from GNU/Linux, the first step is to connect devices at the physical layer. If we are using a laptop and a mobile phone, that connection will be made by means of USB, serial line or Bluetooth technology. If we are using a PDA, the physical link will be established without user intervention. Establishing a point-to-point link layer connection (PPP, *Point to Point Protocol*) is the next step [4]. In Figure 2, we can see the protocol stack used after PPP activation.

One of the most interesting features in the connection process is the access to lower layers. GPRS/UMTS defines a set of AT+ commands to allow the developers to access to the terminal hardware and the GPRS/UMTS network properties. These AT+ commands extend the usual AT commands assigned to control the modem [5].

The application developed in Movicuo, simplifies that process by offering Campus Ubicuo users the possibility to establish a connection by using portable devices and mobile technologies from GNU/Linux. The tool has been implemented using GNOME and GTK+ libraries. Fur-

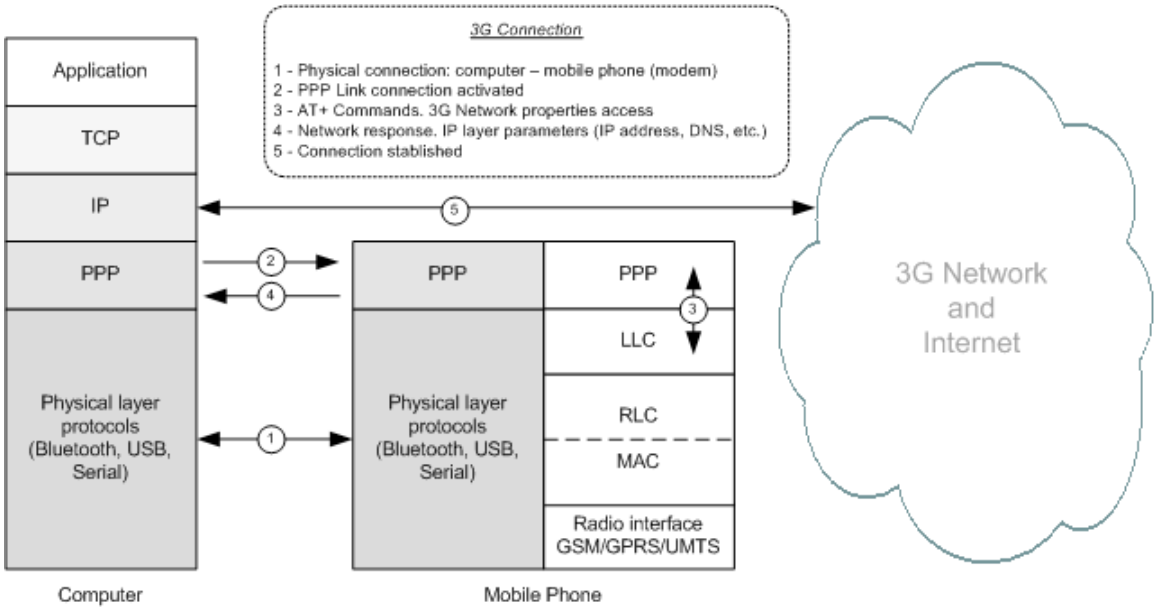


Figure 2: Client-side Protocols for 3G Connection.

thermore, it allows QoS negotiation and offers real-time information about the connection state (see Figure 3). We highlight here the rate/time chart, which is particularly useful, that has been developed using the rdtool <http://oss.oetiker.ch/rrdtool>.

From the research point of view, thanks to this tool, we have analysed 2.5G and 3G connections, studying parameters like *throughput*, *delay* and *jitter*. We have also evaluated the network behaviour depending on the kind of traffic sent and the negotiated QoS level.

2.1.2 GNU/LinEx and PDAs

This section is focused on building a GNU/LinEx distribution for PDA devices. Once developed, this platform will allow us to design advanced communications software for mobile devices. Building a Linux embedded system in a PDA is a complex task. It requires a wide knowledge in operating systems, Linux systems and device specific architecture. PDA uses a class of RISC microprocessors called ARM [6]. This means that the whole software executed in this device must be specifically

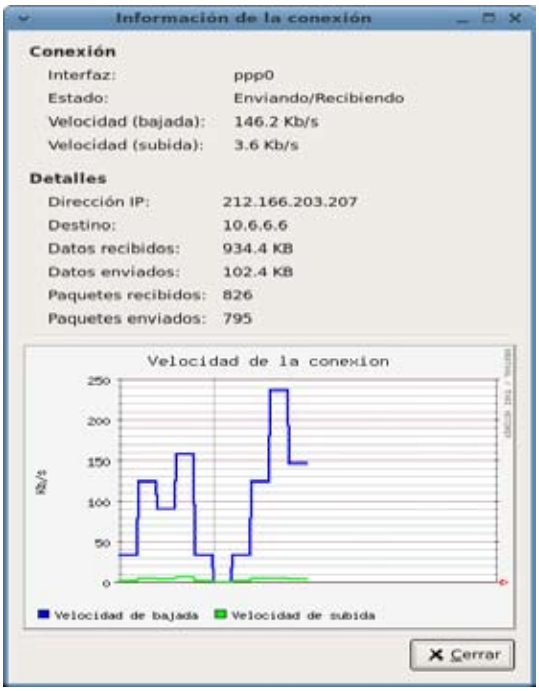


Figure 3: 3G Parameter Information Window.

compiled for that platform. The cross-compiling process consists in creating executables binaries for a platform other than the one on which the compiler is running.

Thus compiling an application for the PDA or to build a distribution implies cross-compiling. Figure 4 shows the difference between a file compiled for x86 architecture and a file compiled for ARM architecture. In this example, the GCC compiler is used.

At present, many projects are focused on Linux system development for embedded devices. Among them

Kernel sources for this microprocessor are *omap-linux* (also needing some patches for this specific device). The cross-compiler used for kernel sources is GCC for ARM (*arm-linux*). This compilation results in an image which will be loaded in the boot process. Besides the kernel, the operating system requires user interactivity tools and the hierarchical Linux filesystem. (We call this structure *rootfs*). Compiling each library and application one by one is not acceptable in time and effort terms, so we use OpenEmbedded to make this task easier. Regarding the

CROSS-COMPILING APPLICATIONS

hello_world.c

```
#include <stdio.h>
int main (void){
  printf ("Hello World!\n");
  return 0;
}
```

x86 architecture compilation

```
# gcc -Wall -o hello_x86 hello_world.c
# file hello_x86
hello_x86: ELF 32-bit LSB executable, Intel
80386, version 1 (SYSV), for GNU/Linux 2.2.0,
dynamically linked (uses shared libs), not
stripped
```

ARM architecture compilation

```
# arm-linux-gcc -Wall -o hello_arm hello_world.c
# file hello_arm
hello_arm: ELF 32-bit LSB executable, ARM,
version 1 (ARM), for GNU/Linux 2.4.3,
dynamically linked (uses shared libs), not
stripped
```

Figure 4: Files Compiled for X86 and ARM Architectures.

are EmDebian <http://www.emdebian.org> or Familiar <http://familiar.handhelds.org>, as well as tools to make easier the cross-compiling process like Scratchbox <http://www.scratchbox.org> and OpenEmbedded <http://www.openembedded.org>. The first of these offers several tools designed to facilitate the process, whereas OpenEmbedded is an environment that simplifies the task of building complete Linux distributions for embedded devices.

In the Campus Ubicuo project we use different PDAs. One of them is HP iPAQ h6340 that incorporates an ARM TI OMAP 1510 microprocessor. Building a GNU/LinEx distribution for iPAQ means accessing each part from which the system is composed: kernel, filesystem and bootloader.

bootloader, we use Uboot because it is supported in the device architecture.

Once we have these three elements (kernel image, rootfs and bootloader) the GNU/LinEx system can be booted. The built PDA platform, which can be extended to other Linux distributions, is the base of other developments and activities in the project.

2.1.3 SARI (Wireless Remote Administration System)

Returning to the architecture shown in Figure 1, Campus Ubicuo has a remote administration system called SARI.

Nowadays, each information system is maintained in a particular way; moreover it should be always available for management tasks. On the other hand, capabilities

of instant messaging technology (SMS, *Short Message Service*) make it especially interesting to manage some critical systems that require to be controlled everywhere and at any time.

There are some applications to administrate a system via SMS, but they have some limitations especially related to security issues. For this reason, we have developed a new robust and extensible system, able to cover the administration needs of modern computing systems. SARI allows remote GNU/Linux commands execution as well as managing subscriptions to the information broadcast system in Campus Ubicuo. These features offer the system administrator the necessary ubiquity in order to carry out administration task. In Figure 5 we can see the interaction between users and SARI.

The main application, or base system, is executed as a daemon. This daemon can be improved by adding different plugins for each kind of technology. At present we have implemented two plugins; one of them in charge of Bluetooth-terminal communication [7] to manage SMS functionality (send/receive) and the other intended to manage the communication via email messages. This way, the daemon executes the corresponding procedures depending on the type of technology supported by the installed plugins. Simultaneously, each plugin follows a set of steps and maintain these steps in a queue. Although the execution of a step is not parallel with respect to the other ones in the other plugins, the global process is concurrent so that different type of technologies can be used at the same time.

Each plugin manages the communication to make it non-blocking. Their implementations have to avoid software blockages if the device or the server does not send a response for a request. Each plugin follows the classic connection stages: establishment, maintenance and closing. The operation of the SARI daemon is similar to the rest of the GNU/Linux services. Its configuration file is located at /etc/sari/sari.conf and can be launched from /etc/init.d/sari. When SARI is running, it allows administrative operations as well as task related to Campus Ubicuo users' subscriptions. To do that, there are several configuration files in /etc/sari where the SARI settings can be tuned.

2.1.4 MeUbicuo

The Campus Ubicuo project has a set of servers where the information is stored that must be managed. This management task has been isolated from the rest of the system, because all the services offered are controlled from these servers. This is why an independent subproject called MeUbicuo has been developed. If some changes are necessary due to changed requirements or to adapt the system to other environments, only MeUbicuo must be modified. SARI is executed on these servers, allowing the users to send and receive information related to Campus Ubicuo subscriptions, as well as some administrative tasks. Databases and data structures necessary to manage the system and the information broadcast system are placed there. This means that when a user is registered for the university news service through SMS and additional news arrives, it will be spread to that user and to the rest

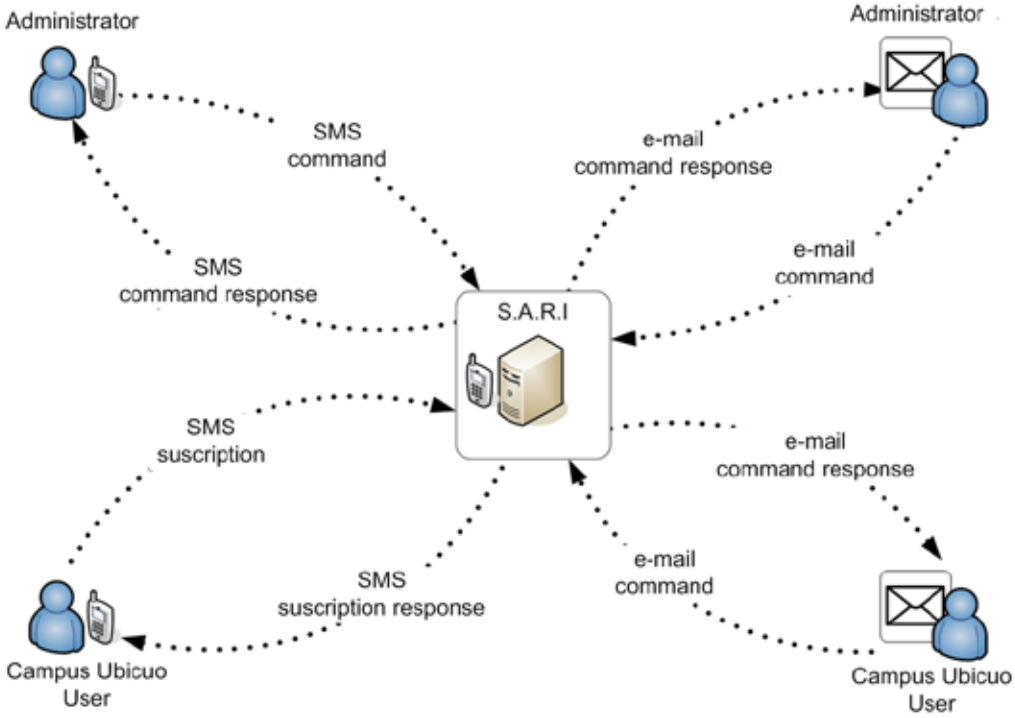


Figure 5: Users-SARI Interaction.

of the users subscribed to that category. In order to allow the users to access the information, a web portal has been developed. This PHP+MySQL application is used by the users to register in the system and to modify or configure their preferences, for example subscribed services or access technology. This method complements the SMS subscription and management service.

3 Research Tasks

Campus Ubicuo is not only a development and innovation project. It has also allowed carrying out research tasks related to mobility for IP networks or wireless transmission interference. These results are briefly presented in this section.

3.1 IP Mobility. Handover Analysing and Optimization

Nowadays, IP mobility is one of the most interesting research topics in the context of networking. Although several approaches have been proposed to deal with mobility, most of them are based on the Mobile IP protocol [8].

Proposed by IETF, this protocol allows mobile nodes to change its point of attachment without losing its ability to communicate. This goal is reached by keeping a permanent IP address in the Mobile Node (Home Address) and another temporary IP address (CoA, Care-of Address) that is valid and routable at the Mobile Node's current point of attachment in a Foreign Network. Handover is one of the costlier processes in Mobile IP. This occurs with mobile node movement, when it changes its point of attachment to the Internet. Some approaches to reduce movement detection latency are based on layer 2 information, so are faster than layer 3 ones. These solutions have an important disadvantage because they restrict the movement among heterogeneous networks due to layer 2 access technology dependence. We consider two main components in order to evaluate layer 3 handover time T_H:

$$T_H = T_{DM} + T_{REG}$$

where T_{DM} is the time interval necessary to detect migrations and T_{REG} is the time interval used to configure the new CoA and register it in the home network.

The handover analysis in Mobile IP has led us to develop a movement detection algorithm called FDML3 (Fast Detection Movement Level 3), that departs from the work done in [9], modifying the original movement detection algorithm proposed in [8]. The handover process in the original standard is based on consecutive unsolicited Router Advertisements (RA) losses. This method is not dependent on the link layer. It depends on the frequency of unsolicited RA. This frequency has less influence in FDML3 algorithm because the first RA loss is used to inform that a handover has happened.

In order to analyze and validate the new algorithm results, as well as to compare it with other detection movement methods, we have used the OMNET++ simulator <http://www.omnetpp.org>, where our proposal has been implemented. Figure 6 shows a chart where 8 movements (handovers) are compared.

The obtained results show that detection movement in the handover process is reduced by up to 25% with respect to original algorithm proposed in the Mobile IPv6 standard. Even so, in some cases, handover time is not reduced because of the time interval between unsolicited RA. If this interval is short, the global handover time is not significantly improved.

3.2 Interference Analysis

Campus Ubicuo has also allowed to make an analytical study and an evaluation of interference in relation to

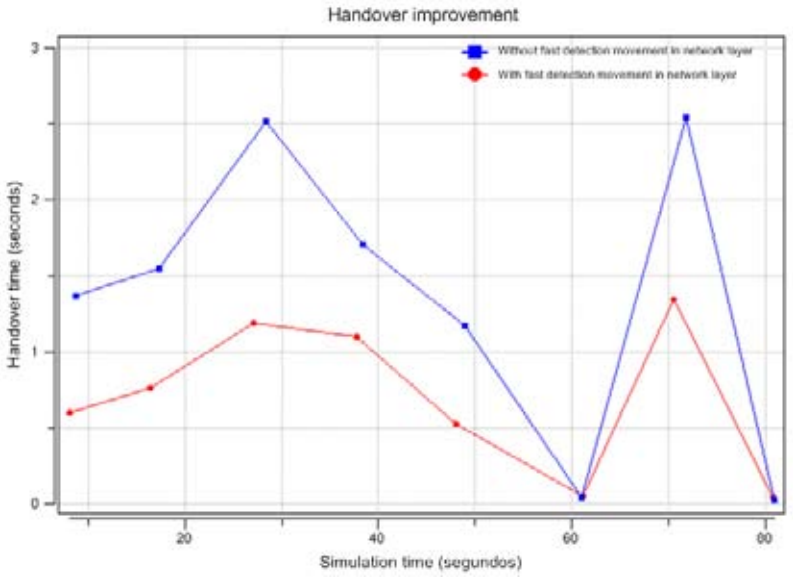


Figure 6: Handover Time with and without FDML3.

different wireless communications.

In this work we have analysed how a wireless technology affects others in a nearby environment. We have analysed how these interferences influence in the performance of each one of the other signals. The research work has been carried out from the point of view of a network administrator, whose job it is to maintain the network operating correctly and to solve the problems that can happen. Some of these problems are caused by interference.

This way, we have been able to prove via laboratory experiments, how wireless networks could be affected by common elements like microwave ovens, wireless phones, radio frequency remote controls and, in general, those devices operating in the ISM (*Industrial, Scientific and Medical*) frequency of 2.4 GHz. This range is internationally reserved to be used in a non-profit environment without the need of a license. For example, most of wireless technology like Bluetooth or Wi-Fi operates in the ISM frequency. Due to this research work, a network administrator is able to know the cause of a performance decrease in a given time and can act to remedy that situation. To carry out this research work, we have used a spectrum analyser that allows us to know how many wireless devices are active, the channel selected by each one of them to operate and some other useful information.

4 Conclusions and Future Work

In the last few years, we have seen an incremental change in needs of users of communication technologies. Mobility services are more and more in demand and last generation data networks are participants in this change. In this situation, the system developed in Campus Ubicuo offers advanced communications and mobile services to its users.

One of the fundamentals of this project is free software, a basic principle upon which we have developed the proposal. This kind of project, managed in the context of an agreement between university and the business field, will allow both GITACA research group and SADIEL S.A the technological transfer to other research groups or companies. Thanks to the advantages of free software, they will be able to profit from generated innovation because our research results will be in the public domain via the project's homepage <http://gitaca.unex.es/cubicuo>.

Although the project is near to conclusion, there is one of the most attractive points still unfinished: Campus Ubicuo deployment in other organizations separate from the university. The use of Campus Ubicuo in three different environments, where it could be directly applied, is now being analysed: hospitals, secondary education centres and tourist services of a local council. In secondary education centres and hospitals, this project can be used to make easier and faster every administrative process for students and patients. In relation to tourist services, Campus Ubicuo can be a support to provide advanced mobile communications services to tourists; for example using a PDA or a mobile phone.

References

[1] I. Herraiz, J. J. Amor, A. del Castillo. "Libre software for research". FLOSS International Conference. Jerez de la Frontera, Spain. March 2007, pp. 97-105.

[2] J. Carmona Murillo, J. L. González Sánchez, A. Gazo Cervero, L. Martínez Bravo. "*MOVICUO: Comunicaciones móviles y software libre para la ubicuidad*". *III Jornadas de Software Libre de la Universidad de Cádiz*, Cádiz, Spain. April 2006, pp 45-58.

[3] J. Perez-Romero, O. Sallent, R. Agustí, M. A. Díaz-Guerra. "Radio Resource Management Strategies in UMTS", Wiley, 2005. ISBN: 0470022779.

[4] C. Andersson. "GPRS and 3G Wireless Applications". Wiley. 2001. ISBN: 0471414050.

[5] ETSI TS 07.07. "Digital cellular telecommunications system (Phase 2+); AT Command set for GSM Mobile Equipment (ME)", 2003.

[6] C. Hallinan. "Embedded Linux Primer". Prentice Hall, 2006. ISBN: 0131679848.

[7] A. Huang. "The use of Bluetooth in Linux and location aware computing". Master thesis, Massachusetts Institute of Technology. Cambridge, MA. 2005.

[8] D. Johnson, C. Perkins, J. Arkko. "Mobility Support in IPv6". IETF RFC 3775, June 2004.

[9] N. Blefari-Melazzi, M. Femminella, F. Pugini. "A layer 3 Movement Detection Algorithm Driving Handovers in Mobile IP". Wireless Networks 11 (3), pp. 223 – 233. 2005. Springer Netherlands.

The Case of the University of Cádiz's Free Software Office Among Spanish Universities

José-Rafael Rodríguez-Galván, Manuel Palomo-Duarte, Juan-Carlos González-Cerezo, Gerardo Aburruzaga-García, Antonio García-Domínguez, and Alejandro Álvarez-Ayllón

During the first years of the twenty first century, executive bodies at Spanish universities were becoming increasingly interested in using free software as a means to their ends. We will describe below the case of the Free Software Office of the University of Cádiz, showing its structure and the problems that such an organization must deal with. Later on, the main projects we have been focusing on since the Office's inception in 2003 will be enumerated, pointing out similarities between other offices and secretariats akin to ours.

Keywords: Free Software, Free Software Conferences, Free Software Popularization, University.

1 Free Software in the Spanish University System

Free software is deeply rooted in the academic world. Even before there was an understanding of the concept itself, scientists and engineers at universities (especially

Authors

José Rafael Rodríguez-Galván works as a teacher in the Department of Mathematics in University of Cadiz. Since 2004 he has been the chair of the OSLUCA (Free Software Office of University of Cádiz), organizing several projects including I, II and III Free Software Conference at the University of Cádiz, and I FLOSS International Conference (FLOSSIC 2007). He has been invited as speaker to many meetings and symposiums related to free software and the University. He is also member of the UCA researching group FQM-315, where he develops his research in numerical simulation of equations for partial derivatives applied to fluid mechanics. <osl@uca.es>.

Manuel Palomo-Duarte received a degree (M.Sc.) in Computer Science from the University of Seville (2001). He works as a full-time teacher in the Department of Computer Languages and Systems in University of Cadiz, where he teaches subjects related to operating systems and videogame design using free software. He is also Erasmus Coordinator for B.S. Degrees in Computer Science (*Ingeniería Técnica en Informática de Sistemas*). He is a member of research group "Software Process Improvement and Formal Methods", and he's pursuing his PhD. about quality in BPEL web service compositions. Since he joined University of Cádiz he has collaborated with the Free Software Office, mainly developing conferences: III Free Software Conference in University of Cádiz (JOSLUCA3) and I FLOSS International Conference (FLOSSIC 2007). <osl@uca.es>.

Juan-Carlos González-Cerezo is a Computer Technical Engineer graduate of the University of Cádiz. He is "*Premio Extraordinario de Fin de Carrera de la Diplomatura de Informática*" 1990/1991 by the University of Cádiz, and the second National Award for the End of the Studies in Computing Science B.S. Degree, from the Ministry of Education. He has been a programmer in the Computer Services of the University since 1992. He is now participating as a technician in the Free/Libre Software Office in the UCA, while he is continuing his studies in Computer Engineering. <osl@uca.es>.

Gerardo Aburruzaga-García is a coordinator of the Computing Area of the University of Cádiz, and he has been an associated teacher in the Computer Languages and Systems Department since 1989. He has worked as a programmer and an analyst in the computer services of the UCA, where he commenced in 1988. He now has technical responsibility for the Free/Libre Software Office. As a teacher, he has taught Logic Stands, Programming II, Programming Methodology and Technology II, Operating System Administration and Object-Oriented Programming. <osl@uca.es>.

Antonio García-Domínguez is a Computer Technical Engineer (B.S. degree) graduate of the University of Cádiz (2006), where he is continuing his studies in Computer Engineering. He is "Mención Especial en el Premio Nacional de Fin de Carrera" 2005/2006. He submitted his final year project "Post-procesador y visor de demostraciones para el sistema de demostraciones ACL2", and in the summer of 2007, he participated in the Openlab programme project "Grid-based financial software" at CERN. He is now taking part in the development of a framework for BPEL web services composition testing. <osl@uca.es>.

Alejandro Álvarez-Ayllón is a Computer Technical Engineer (B.S. degree) graduate of University of Cádiz (2007), where he is continuing his studies in Computer Engineering. His final year project was a forge, built in Zope/Plone, specially oriented to the university community, and it is being used as the official project repository by the Free/Libre Software Office of UCA. He is an intern of the Free/Libre Software Office of the University of Cádiz, an active collaborator of the QtOctave project, and the main developer of Hispaciencia.com. <osl@uca.es>.

American universities) worked according to a knowledge exchange model which could be seen today as an open source developer community. Later on, in the middle of the 1970s, as the circumstances changed to what they're like today, the different and isolated initiatives which could be labeled as "free" or "open" originated from academia and research centers. Such is the case of TEX, for example, and, in some way, of UNIX itself. The latter was published with a license which in practice allowed for its free use for academic purposes and it became the source of the BSD operating systems at the University of Berkeley.

The creation of the free software concept itself is attributed to people like Richard Stallman and Linus Torvalds, who had strong links to the academic world.

Though the Spanish academic system stayed unaware until the 1980s of this phenomenon, it saw a gradual expansion of free software in the later decades of the twentieth century, and of the GNU toolset in particular. Their remarkable quality, and their availability at no monetary cost in FTP servers (such as the well-known *sunsites*) sped up their adoption by IT departments at Spanish universities. Likewise, some groups of teachers started to get acquainted with free software and using it to support their research and teaching. Due to the quality of the GNU/Linux operating system, along with its increasing popularity and maturity and the ease of configuration and installation that the several distributions provided, it started to replace the proprietary UNIX systems commonly seen on university servers and in data centers.

In the first years of the twenty-first century, there was a substantial free software community in Spain, made up largely from individuals and groups associated with the university system, either isolated or as a part of some association, but with no institutional support. It was already by then possible to imagine free software making the leap to the desktop, having office suites which could rival commercial solutions. Upon this foundation, large scale political initiatives at public administrations to use GNU/Linux systems and free software such as the *Junta de Andalucía* (main governing body in the Andalusian region) or the *Junta de Extremadura's* efforts saw the light.

Executive bodies at public universities considered that this work couldn't be overlooked, in accordance to their natural duty of transferring higher knowledge and the moral senses of solidarity and freedom. Of course, the chance of reducing the costs associated with (frequently restrictive) software licenses was a powerful incentive. In this way, the first institutional announcements supporting free software were published, and the first university organizations dedi-

- Universidad de Alicante: **COPLA** <http://copla.ua.es/ >.
- Universidad Autónoma de Barcelona: **GNUAB** <http://www.gnuab.org/ >.
- Universidad de Barcelona: **gclUB** <http://gclub.ub.es/>.
- Universidad de Cádiz: **OSLUCA** <http://softwarelibre.uca.es/ >.
- Universidad Carlos III de Madrid: **LUC3M** <http://luc3m.uc3m.es/>.
- Universidad de Castilla-La Mancha: **CRySoL** <http://crysol.inf-cr.uclm.es/>.
- Universidad de Deusto: <http://softwarelibre.deusto.es/>.
- Euskal Herriko Unibertsitatea: **itsas** <http://itsas.ehu.es/>.
- Universidad Europea de Madrid: **GLUEM** <http://www.gluem.net >.
- Universidad de Huelva: **OSLUHU** <http://cibercomunidades.net/uhu/osluhu/>.
- Universidad Jaime I: **Software Libre UJI** <http://www.swlibre.uji.es>.
- Universidad de A Coruña: **OSL-UDC** <http://softwarelibre.udc.es/>.
- Universidad de La Laguna: **SSL-ULL** <http://ssl.ull.es/>.
- Universidad de Murcia: **SOFTLA** <http://www.um.es/atica/softla/>.
- Universidad de las Palmas de Gran Canaria: **OSL** <http://www.softwarelibre.ulpgc.es/ >.
- Universidad Politécnica de Cataluña: **CPL** <http://www.cpl.upc.edu>.
- Universidad Politécnica de Valencia: **poLinux** <http://www.polinux.upv.es>.
- Universidad Pontificia de Comillas: **linuxec** <http://linuxec.upcomillas.es/>.
- Universidad de Sevilla: **SOFLA-US** <http://solfa.us.es/>.
- Universidad de Valencia: **LinUV** <http://www.uv.es/LinUV/>.
- Universidad de Valladolid: **SOLEUP** <http://soleup.eup.uva.es>.

Figure 1: Offices, Secretariats and Free Software Associations in the Spanish University System (source: "Iris Libre" meeting, November 2007).

cated officially to its development and popularization appeared. This was such the case with the official free software support announcement at the UAM (Universidad Autónoma de Madrid), and the inception of the Free Software Office at the Universidad de Las Palmas de Gran Canaria.

This is the environment in which our own Free Software Office (*Oficina de Software Libre de la Universidad de Cádiz*, OSLUCA from here on), and in later years other entities with differing levels of institutional support, were created (see figure 1). The OSLUCA case will be illustrated below as a paradigm of the work that these entities have been conducting, pointing out both the most common tasks, such as organizing conferences or developing local GNU/Linux distributions, and more specific work, as the interoperability framework which was approved in an executive meeting.

In the last few years, the academic free software initiative has experimented with participation in the discussion groups normally used for knowledge exchange and collaboration. This is the case of the Iris-Libre workgroup, part of the Rediris community, and of the CRUE-TIC SL *Subgrupo de Trabajo en Software Abierto* (Open Software Work Subgroup), belonging to the *Conferencia de Rectores de las Universidades Españolas* (the Spanish universities' rectors association). Coordinating efforts between different universities is crucial to avoid common mistakes and to achieve otherwise impossible objectives.

2 The OSLUCA Case

In 2003, the University of Cádiz was home to some groups of teachers who, being users of free software, had organized varied initiatives, such as summer courses in this context, and were related to the Spanish free software community. Additionally, the IT department had rebuilt most core University services (such as mail or Web servers) upon free software, passionately defending their approach.

Assisted by some of these people and learning of the favorable situation in Andalusia and of the first initiatives which were being born in Spanish universities, the then recently formed executive body pledged its support for the use of free software in the University of Cádiz. The IT General Management supported this pledge as a critical course of action, in which some of the IT staff and teachers most active in the free software circles became involved, starting some work on the area. These were:

- Creation of an information bulletin which was distributed internally around the University.
- First initiatives towards the popularization of free software, among which Richard Stallman's visit in July 2003 stands out.
- Development of a website, a forum, several mailing lists, and so on.

On 15th March 2004, the Executive Council of the University of Cádiz, its highest ranked executive body which decides overall strategies to be followed, approved officially a declaration of support of free software (included in the UCA Official Bulletin #9), and formally established the *Oficina de Software Libre de la Universidad de Cádiz*.

This ambitious declaration acknowledges that free software is part of the means to the ends of the UCA, and states the role of the OSLUCA in "promoting the use of applications and computing resources based on free software in the academic community", particularly so in "taking the necessary measures" to "guarantee the non-discrimination" of the users and "boosting the use and development of free software" in the University, backing the training in the use of free tools, their use for teaching in computer classrooms, for management and research, and the publishing of any derived material with free licenses, among others.

The OSLUCA comprises several teachers, IT staff and collaborating students within the following structure:

- The *Free Software Office Director* is a teacher, charged with the management of the office. Though the position has included since its inception a reduction of teaching work, recently it has been considered within the Secretariat Management category, in the same way as some free software secretariats in other Spanish universities.
- The *Technical Director* is a UCA IT staff member highly experienced in the use of free software who integrates this work in his own routine. There's also another IT staff member which does important support work for the OSLUCA.
- Since 2005, the OSLUCA has had its own facilities. At that time a specific student scholarship for collaboration with the Office was created. There have been also other more specific scholarships, for example, for working on projects which were of interest to the University.
- Among the teachers at the UCA, some of them have been very participative, undertaking critical tasks in the OSLUCA.

3 Work Performed

With the backing of the University's executive body, the OSLUCA had an encouraging outlook, but also heavy responsibilities and many hardships to endure, most of which have been common to all Spanish universities. In view of this, various initiatives have been undertaken, which could be classified under the following general courses of action:

1. Non-discrimination of university members who wanted to use free software. Starting with Web mail or database queries, a university must offer services of all sorts to several thousand members. For a long time, it was assumed that all of them would use the most common environment (Windows machines running Internet Explorer), usually leading to problems for free software users who wanted to use these services. Similarly, the ignorance of the importance of open standards led to problems in exchanging

documents inside the University. This was the case of most office documents, saved under Microsoft Office's closed proprietary formats.

2. *Popularization and training on free software tools, and in the collaboration and publishing of free content*. In the Spanish university system, just like in the rest of society, daily work is usually accomplished with proprietary tools. Introducing free software implies a change which leads to initial rejection, which can only be overcome through training, by helping migration and focusing on its unique strengths. It is here where holding conferences, courses and all sorts of activities which can expand public knowledge about free software, its philosophy, advantages and recommended tools plays a major role.

3. *Technical support and installation of free software for its use in teaching, research and management*. Initially, people who brave a migration to free software have to face some extra work which offers few immediate returns. Providing them with support in the same level that the one offered by commercial software is thus critical to avoid discrimination. In the same way, all projects aimed towards using free software in common tasks in the university, and their conception, must be supported.

4. *Developing projects of strategic interest*. As part of our role in boosting the use of free software for common tasks in the University, frequently we need to get involved in highly important projects which need help to reach fruition, in the form of documentation, development and so on. This would be the case, for example, of free alternatives to widely used applications in teaching, research or management. Interestingly enough, the proprietary software licenses, often abusive, happen to be a quite legitimate reason for the development of these alternatives.

From these general courses of action, we will describe below some of the most representative efforts that have been executed in the OSLUCA. It is important to note that some of them have been common to several universities. We will comment on this in every case.

3.1 Supporting Free Software Usage in several Services at the UCA

The OSLUCA has offered support for free software use in the UCA's services. Due to its high quality, most of our services have been built on it for a long time (Web servers, proxy, webmail, and so on), as in most other universities.

In other cases, the migration to free software has been more recent (with the OSLUCA providing support to the extent of its abilities). This is the case of the University of Cádiz portal, which uses the content management service Plone (based on Zope), and of the course management system, now using Moodle. Moodle was introduced in 2005 as an alternative to the proprietary system which had been in use until then (incurring a large cost), being a clear example of the economic benefits of finding free alternatives.

On other occasions our staff has undertaken directly the setting up and tuning of services. This was the case of the free Internet access points (using GNU/Linux systems), the over 240 GNU/Linux-based laptops which can be borrowed at the library, and the increased number of classrooms with dual boot (Windows and GNU/Linux) machines.

3.2 Creating the UCA Institutional Information Exchange Regulation

On 27th September 2004, the Executive Council of the University of Cádiz approved the "Institutional Information Exchange Regulations at the University of Cádiz", proposed by the OSLUCA (UCA Official Bulletin #15, page 63).

These regulations mandates that "every document published by a body part of the UCA officially directed to some of its members must be saved in an open format, whenever a suitable one exists, or at least must include a version saved in an open format, independently of the medium used for its transmission: electronic email, website, etc.". Lastly, they also indicate that it will require the IT staff to keep an updated list with the open formats for the different kinds of documents to which institutional documents have to conform.

The UCA regulations were the first document of its kind that was published in Spanish universities and public administrations, and it was among the first worldwide. In fact, four years later, there are few similar initiatives in our surroundings.

During 2005, the IT department and the OSLUCA had to undertake the creation of an open format taxonomy and selecting applications which were suitable for use in the UCA. The publication in September 2005 of the PDF/A (ISO 19005-1) standard, and in November 2006 of the OpenDocument (ISO 26300) standard, allowed us to rely on proper standards for their use in office work. Therefore, during 2006 and 2007 a training program for management personnel at the UCA was instituted. This would become a compulsory training course intended to become the starting point for the enforcement of the published regulations. Such a revolutionary initiative must somehow break through the deeply ingrained habit of using Microsoft Office within the university, thus causing us to undertake some additional work which would lead to its eventual success.

3.3 Guadalinex_UCA

Several Spanish universities have developed their own GNU/Linux distributions. In fact, the Metadistros project (upon which many distributions, like some of the Andalusian ones, were based) was born as part of a project to ease the creation of academic distributions in the Free Software Office of the University of Las Palmas de Gran Canaria. There was a rather heated debate about this in the OSLUCA. We list several advantages:

1. Having one more way to promote free software in our university. If the distribution allows for a "live" boot from disk, this would help users to try it out with little hassle.

2. Being able to ship in the distribution some additional recommended applications, and extra content (documentation, for instance) so they can be tested out in our

university. We could also configure the distribution so the UCA's resources could be used straight away.

3. Creating a well-defined system for which we could offer technical support, while gaining some insight into creating GNU/Linux distributions, something that could be useful in certain circumstances. On the other hand, there are some drawbacks:

1. Creating a distribution requires a constant effort to keep it updated release after release.

2. With all the available distributions, creating one more might be a recipe for confusion.

In the end, it was decided that an adaptation of the Guadalinex distribution (the first and only not from the *Junta de Andalucía*, though with its full support) called Guadalinex_UCA would be created. We emphasized that it wasn't a new distribution, but a specialization of an existing one, and that it was also a way to support, as an Andalusian university, the *Junta de Andalucía's* efforts in supporting free software.

Guadalinex_UCA caught the eye of many people and with several thousands of copies distributed it proved that this sort of initiative was possible with the limited resources of a free software office. However, it didn't have quite the impact that we expected, and keeping it updated required considerable effort. Nowadays, we are investing that effort into creating a selection of recommended software at our university. Our next objective will be to spread the word about these programs and help in their distribution through metapackages and compilation CDs. This includes both extra CDs for Guadalinex or for other operating systems including MS Windows. Publishing CDs with free software for Windows to boost their popularity is something that has been promoted in several free software offices around Spain.

3.4 Free software in Teaching

With some exceptions, the use of free software during teaching in the UCA is not widespread. Sometimes, the reason is the lack of proper alternatives to the applications used for teaching, but in most cases it's simply due to either the ignorance of the existence of such alternatives or the lack of motivation for doing such a migration.

Therefore, the OSLUCA has cooperated with the teaching body in classifying and spreading the available alternatives, creating training programs for the most appropriate ones and improving them when needed.

Some examples include:

1. Boosting the use of the *Maxima* application in the Mathematics Department by a program-contract, and the organization of a training course for teachers in 2006.

2. Participating (since 2007) in the development of *qtOctave* (a high-level language intended for numerical computations), a GUI for Octave, one of the applications which could become a viable alternative for its use in many subjects from different departments of our university.

3. The R-UCA project (2007), which simplifies the implementation of the R statistical package as the standard for teaching and research in this area. The project focuses on several courses of action, which include creating teaching materials (using free licenses), training and helping teachers, participating in the translation and development of existing interfaces, and so on. This is also a critical project, since using a free and high quality statistical package such as R would allow the University to avoid the abusive proprietary software pricing policies.

3.5 Other Work

Among our other initiatives, we should point out our efforts in promoting the creation of dissertations and masters' theses under free licenses. For instance, we have developed our own software forge for our students' projects. We have also participated from its very first edition in the *Concurso Universitario de Software Libre,* where a prize is awarded for the best free software project developed by a student of a member University. This contest was created in 2006 by students, teachers and several entities related to the University of Seville. It has attained considerable success. Starting with the 2007/2008 academic year, the UCA also organizes a local contest, in which a prize is awarded for the best project created by UCA students. Our aim is to boost participation in the contest.

4 Workshops and Conferences

Just like most other Spanish free software offices, the OSLUCA has invested a great deal of effort in popularizing and promoting free software through many conferences, workshops and seminars. Some of them stand out from the rest:

4.1 I Jornadas de Software Libre de la Universidad de Cádiz

This conference took place on 14th and 15th April 2004, and served as the first public appearance of the OSLUCA at the University. Some of the participants included Jesús-María González-Barahona (from the Libresoft group of the University Rey Juan Carlos), José-María Rodríguez-Sánchez (IT General Director of the Information Society initiative at the *Junta de Andalucía)*, and Roberto Santos (Vice-president at Hispalinux). It must also be noted that the second day was dedicated to the Zope platform upon which the University's portal is based.

4.2 IV Jornadas Andaluzas de Software Libre

Taking place on 5th and 6th November 2004 at the *Escuela Superior de Algeciras* (Algeciras High Technical College), they were run jointly by the OSLUCA and the ADALA and CAGESOL associations. The talks included those of Álvaro López-Ortega (GNU project developer), Antonio Larrosa (KDE developer) and Juan Conde, from the *Consejería de Innovación, Ciencia y Empresa* (Innovation, Science and Business Council) of the *Junta de Andalucía*.

4.3 II Jornadas de Software Libre de la Universidad de Cádiz

Held from 6th to 8th April 2005, it was organized along two main themes: free software in science, and free software in business.

On the first part, the talk about the ethical aspects of free software by Ricardo Galli (teacher at the *Universitat de les Illes Balears* and main speaker of the *Free Software Foundation* in Spain) and the talk about supercomputation by Antonio Fuentes-Bermejo (member of IRISGrid, Iniciativa Nacional de GRID, from the Rediris network, the National GRID Initiative). Juan-José Hierro (from the Morfeo project at Telefónica I+D), Isidro Cano (supercomputing director at HP) and Javier Viñuales-Gutiérrez (Yaco Sistemas) participated in the other area.

Moreover, the conference hosted workshops in computer classrooms for the first time. One of them focused on the creation of GNU/Linux distributions based on Guadalinex, and another on Blender, the excellent (and free) 3D modeling, animation, rendering and post-production application.

4.4 III Jornadas de Software Libre de la Universidad de Cádiz

This conference, held on 20th and 21st April 2006, represented an important leap forward in quality. For the first time, a formal call for papers and peer review process for participation in the Conference was followed, with all work published afterwards in the conference's proceedings under ISBN. Furthermore, the Computing Languages and Systems department gave a helping hand in the organization of the event. More than 200 CDs with the conference's proceedings and 100 free documents (manuals, books, tutorials, and so on) were distributed.

Among the speakers, Ismael Olea (Spanish Linux Documentation Project), Juan M. Rocha-Ramos (administrator of the *Conocimiento Libre* forge at the *Centro Informático Científico* de Andalucía) and Juan-Jesús Ojeda (Guadalinex developer) were included. Serving as a prelude, a seminar on "Intellectual property, free software and the university system" was held. César Iglesias (Díaz-Bastien & Truan, Attorneys at Law) and Malcolm Bain (member of the Cátedra de Software Libre at the UPC), among others, attended the seminar.

4.5 FLOSS International Conference

The FLOSS International Conference was organized jointly by the "Software Process Improvement and Formal Methods" research group and the University of Cádiz Computing Languages and Systems department. The *Escuela de Negocios* de Jerez (Jerez Business School), a part of the Jerez City Hall's *Delegación de Formación y Empleo* (tasked with boosting employment and training in Jerez), also cooperated.

This conference took place in the Jerez Social and Communication Sciences Faculty, part of the University of Cádiz, between 7th and 9th March 2007.

Participation was excellent, with over 30 speakers from several countries, and more than a hundred attendants, with researchers, teachers, technology workers and university students.

The papers presented had a strong scientific focus, with topics varying from e-learning, web accessibility, automatic translation, software development, professional 3D modeling, supercomputing and artificial intelligence. Among the most important speakers were Rüdiger Glott (from the UNU-Merit University), Alberto Barrionuevo (FFII vice-president), Juan-José Domínguez-Jiménez (SPI&FM group) and Juan José Amor (Libresoft group) were included.

All papers were included in the proceedings, which were published by the University of Cádiz Editorial Service with its own ISBN, under a free license. The proceedings were also distributed in a compilation CD with over 200 books, manuals and tutorials about free systems. These were given out to all attendants and may be freely downloaded from the event's website.

At the same time, jointly with the Jerez Business School, a seminar on the opportunities which are presented by free software in business was held. Several IT firms presented the varied solutions that they offer to their clients, ranging from office suites all the way up to clusters and distributed computing, and including ERP systems, groupware, CRM applications, among others.

5 Conclusions

Although the Free Software Office of the University of Cádiz was created only recently, it has already achieved important goals, being a Free Software regional reference institution.

Future goals include an increased adoption of free software solutions in every aspect of the university's daily work: management, teaching, researching and collaborations with external institutions.

On Understanding how to Introduce an Innovation to an Open Source Project

Christopher Oezbek and Lutz Prechelt

This article was previously published in the Proceedings of the First International Workshop on Emerging Trends in FLOSS Research and Development. FLOSS '07. ISBN: 0-7695-2961-5. Digital Object Identifier: 10.1109/FLOSS.2007.11. It is reproduced with kind permission of IEEE and the author.

We propose to research the introduction of Software Engineering inventions into Open Source projects (1) to help researchers with creating opportunities for evaluating their tools, methods and process designs in real-life settings, and (2) to help Open Source projects with improving their processes based on state-of-the-art knowledge. Such research will go beyond diffusion and dissemination of inventions to active introduction, and thus increase the chances of adoption. We will discuss the research approach, our preliminary insights, limitations of the approach, and why researchers interested in evaluating their own inventions should be interested in this research.

Keywords: Empirical Methods, Innovation Adoption, Open Source, Software Engineering Invention.

Authors

Christopher Oezbek received the *Vordiplom* in informatics from *Universität Karlsruhe* (2002), the MS in computer science from Georgia Institute of Technology in 2004 and is currently pursuing his PhD at *Freie Universität Berlin*. His research interests include Open Source development processes, source code documentation and API usability. He is a member of ACM and GI (*Gesellschaft für Informatik*). <oezbek@inf.fu-berlin.de>.

Lutz Prechelt is full professor of Informatics at the Freie Universität Berlin since 2003. Until 2000, he worked as senior researcher at the School of Informatics, University of Karlsruhe, where he also received his Ph.D. in Informatics in 1995. In between, he was with abaXX Technology, Stuttgart, first as the head of various departments, then as Chief Technology Officer. His research interests include software engineering (using an empirical research approach), measurement and benchmarking issues, and research methodology. His current research topics revolve around open source software development, agile methods, and web development platforms. Prechelt is a member of IEEE CS, ACM, and GI (*Gesellschaft für Informatik*) and is the editor of the Forum for Negative Results (FNR) within the Journal of Universal Computer Science (J.UCS). <prechelt@inf.fu-berlin.de>.

1 Introduction

Most software engineering research produces technology such as tools, methods, or processes to be applied during the construction of software systems. It has been gradually understood that the empirical evaluation of such inventions is necessary to judge research progress and generate acceptance outside of academia [25][28].

There are two classic scenarios for how to conduct such empirical evaluations: First, there is the laboratory trial, often in the form of controlled experiments with student subjects. Such studies are difficult to set up in such a way that they are sufficiently impartial and realistic (in particular in their choice of task) to be credible—but credibility is what counts [19]. Controlled experiments with professional subjects are harder to set up, but often hardly more credible. Second, there is the industry trial, commonly performed as a case study in cooperation with a company. While such studies are certainly realistic, they have problems too: Cost and risk considerations make it hard to find industrial partners, non-disclosure constraints make it hard to fully describe the setting and results, and company idiosyncracies often make it hard to understand generalizability.

For many (though not all) evaluation purposes, some researchers consider observational studies in the context of Open Source Software (OSS) projects to be a third approach and one with almost ideal properties in many respects: Credibility can often be high, they are easy to observe, publication constraints hardly exist, risk considerations are more relaxed, and corporate cost considerations are replaced by (mere) group willingness hurdles.

Unfortunately, OSS projects are not interested in studies, they are interested in developing software. So, performing a study first requires to make the project adopt the invention in its normal work. However, as anybody knows who has ever tried to get any group of people to adopt an invention (that is, to introduce the invention as an innovation), this is rather difficult. So, rather than letting a long row of researchers individually attempt, fail, attempt, fail, get frustrated, and give up, we suggest to make the adoption process itself the subject of research in order to provide these researchers with a proven methodology for introducing an invention to an OSS project.

Here the term *introduction* is used to signify the planned initiation of an adoption process within an organization or

social system. *Adoption* then can be seen as the turning point where inventions become innovations that are actively used by individuals [7]. *Introduction* contrasts well with *diffusion*, which carries more passive connotations, and *dissemination*, which does not go beyond distributing information or resources related to an invention.

From the researcher's point of view, combining active introduction with OSS projects has several advantages. In contrast to industry settings, the public visibility of most of the working process, artifacts, and communication as well as the openness for outsiders to contribute to these projects allow the researcher to both capture and influence the project to a much larger degree. In contrast to dissemination and diffusion, the researcher can (1) observe the adoption and use of the invention as it happens rather than performing post-hoc analysis, (2) tailor the invention to the particularities of the project and repair problems that often plague early versions of inventions on the spot, and (3) choose the project such as to maximize the insights gained.

From the point of view of the OSS community, such research increases their chances for benefitting from software engineering improvements, given the fact that conventional approaches to managing software process improvement such as CMMI [5], even approaches specialized to OSS [8], do not explain how the actual introduction of the improvements should be conducted, and traditional key success mechanisms such as management commitment and support [24] are unlikely to work.

The rest of the paper presents our research approach for gaining insights into the introduction of inventions in OSS projects as well as our preliminary results for the following research questions:

1. How to select target projects suitable for introducing software engineering inventions.
2. How to approach a project to offer an invention.
3. How to interpret reactions and make strategic and tactical decisions based on them in the course of the adoption process.
4. How to phase out involvement and exit the project.
5. How to obtain evaluation result data during and after the introduction.

2 Research Approach

To develop an understanding of the introduction of inventions, we will perform a series of iterative case-studies [27] using action-research methodology [2], i.e., a circular, collaborative process of planning, acting and reflecting. These studies will be performed with three different inventions of different type and with a variety of different Open Source projects. We will not introduce several process improvements in the same project [9] in order to avoid synergies or cannibalization between improvements [11].

Inside each case we will gather qualitative data on action-reaction relationships and recurring patterns (using Grounded Theory data analysis methodology [6]) to obtain an understanding of the key interactions during an introduction effort.

We will work on minimizing risk toward the project and on protecting the autonomy of the subjects [4] by creating an atmosphere of collaboration, involvement and participation between project and researcher, and protecting privacy and confidentiality [3][13]. Even though Open Source projects are very robust against negative influence from the outside, similar precautions must be taken by researchers who evaluate their inventions in projects to ensure proper ethical conduct.

3 How to Choose a Host Project

Choosing an appropriate Open Source project when evaluating a software engineering invention is important to establish a case that is (a) typical enough to generalize to other projects, (b) suitable for the given invention, and (c) has potential for interesting interaction regarding the introduction.

In particular, the project should be Open Source not only by license but also by development style: The project members need to be distributed rather than co-located at a single company site, communication must be public and preferably archived, it must be possible for external newcomers to join the project, and basic processes and tools (such as release process, issue tracker and version repository) should be established. The distribution, observability, and openness ensure that the researcher can study the use of the invention at all, while the presence of basic processes and tools indicates that the project probably fulfils basic professional software engineering standards so that study results may generalize to other software development projects. Fortunately, with the existence of project hosts such as SourceForge these tools and processes are now standard.

Regarding the size of the project a viable middle ground must be found between too small and too large. Small projects with less than three to four developers usually have little interaction, communication overhead, tool usage, and process inefficiencies or are still in the process of establishing basic process patterns. They are thus rather unsuitable for all but the most basic software engineering inventions. Large projects with more than fifty developers on the other hand have quite the opposite problem: They usually have well established processes, so that the "not invented here"-syndrome, explicit opposition, tedious consensus finding, low perceived benefit against the established processes, and high communication overhead might make it impossible for a single researcher to be heard. Accordingly, we suggested to chose a middle-sized project: five to fifty developers of whom at least five have been active during the last few months.

As a last project property, we believe it useful to target a project that has shown an affinity for change (or at least no opposition to it) in the past. In many cases this property will correlate with the openness of the project to accept new members, but it is still beneficial to study the history of in-

ventions adopted by the project; a typical example might be the transition from the CVS version control system to the newer and clearly superior SVN.

To acquire a project somewhat randomly yet within the limitations given above, a project news announcement site like Freshmeat, which aggregates projects independently of their hosting, or a project listing site like SWiK can be used.

Both of these example sites offer the option to visit a project at random from the listing. While SWiK shows all projects that relate to Open Source, Freshmeat's notable limitation is its requirement for projects to run under an Open Source operating system; purely Windows-based OSS projects are not listed.

4 How to Approach Open Source Projects

Some knowledge exists in the literature about how to approach an OSS project [10][26]. Firstly, the concept of "gift culture"[21] suggests that the respect for the external participant and influence s/he carries are correlated to his/her contribution to the project. This raises the question whether the invention itself will be seen as a gift if disseminated to the project. A case study on the effects of offering a source code gift that requires further effort to integrate into the code-base of the project appears to indicate the following: Unless the gift is directly useful for the project and immediately comprehensible to the participants, chances are low that it will be accepted [20]. Thus, we hypothesize that the researcher should expect to spend a considerable amount of work generating these benefits until the invention is accepted and adopted.

Secondly, the researcher needs to decide whether to approach the project by contacting the maintainer and project leaders, individual developers, or by addressing the project community as a whole. Our working hypothesis is that the type of approach should be correlated closely with (a) the degree of independence of each member's adoption decision, and (b) the benefit structure of the invention. We will now explain these factors.

In *Diffusion of Innovations*, Rogers distinguishes three *types of innovation-decisions*: *optional innovation-decisions*, which each member of the project can make individually and independently, *collective innovation-decisions*, which require consensus within the project, and *authority innovation-decisions*, which are made by a small influential group within the project [22].

As an example, consider the adoption of a practice such as "mandatory peer review before committing patches to version control". Such an improvement usually starts as a collective innovation-decision to improve code quality, since a general consensus is needed that every member of the project will submit his or her patch first to a mailing-list for inspection, and thus the whole community should be addressed to promote the adoption. Additionally, it also involves an optional innovation-decision by each member to participate in the review of patches sent by others, and thus can be supported by the researcher by talking to individual developers. As an example of the third kind of innovation-decision and its implications for how to approach the project, consider the introduction of a feature freeze[1] two weeks prior to a release. This decision can be driven by the project leaders and maintainers in an authoritative fashion and supported technically by creating a local branch for the release in the version control system. Individual members can undermine the decision, but they need not take specific action to make it a reality. Thus, the researcher should communicate directly to the project leaders.

The second important property of the invention that affects the approach is the *benefit structure* of the invention offered by the researcher, i.e., the return on investment or relative advantage [22] for each project member in contrast to the return on investment for the whole project. The documentation of the project, for instance, does not provide a high return on time spent for the experienced developer who writes it, yet the information is highly useful for new developers (where they might provide large returns for the project). Inventors often understand the *increasing returns* [1] promised by their invention but tend to overlook that (a) individual project members driving the introduction might not benefit from the improvement sufficiently to compensate for the effort they spend on it and (b) the benefits might be hard to measure or only visible in the long-run.

We hypothesize that the researcher should start the approach with those project members who can gain immediate benefits. Instead of asking other project members to perform tasks with a negative bottom line in terms of their personal benefit, those tasks should be performed by the researcher initially. Later on, when the benefits become visible and affect individuals in the project, the researcher will have a much better chance to involve project members and withdraw from these activities.

5 How to Interpret Reactions and Make Strategic and Tactical Decisions

When introducing inventions and novelties of any kind into a social system, the researcher should expect *rejection*, *adoption*, and *reinvention* as ultimate reactions to occur both on the individual and group level [22].

Rejection is the decision not to adopt an innovation. It might occur both actively, i.e. after considering the adoption or even conducting a trial, or passively, i.e. without any consideration at all [22]. Passive rejection, i.e. not getting a response at all, is not uncommon even if the researcher explicitly expresses interest in joining the project [26].

Reinvention occurs if members of the project take up the invention and recast or reuse it in unexpected and unintended ways. Reinventions might prove highly beneficial for the researcher, as they may point to new fields of application for the invention.

[1] In a software release process, a feature freeze is the point from which onwards no new features must be introduced; only defect corrections and documentation are allowed to be performed.

Of course, there is still a lot of room for interaction between the project member, researcher and technology until these ultimate reactions are made. Social science literature provides various models for such discourse such as the theory of fields [12] or network-actor theory [17]. We have chosen to follow the innovation model developed by Denning and Dunham [7]. In this view, the innovation process starts with (1) the sensing of possibilities for change and (2) a vision of what might result from the change. (3) Offering this vision to the affected people (or other units of adoption) and receiving their feedback allows the idea to be shaped into something that can be (4) executed and implemented in concrete terms resulting in a product, process or social improvement. It is only after the invention has been (5) adopted by the desired target population and (6) sustained as a successful novelty that a successful introduction of innovation has occurred. In the setting discussed here, the first two stages will focus more on the tailoring of the existing problem, vision and invention rather than the generation of new ideas and implementation.

6 How and When to Phase Out Involvement and Leave?

Our current working hypothesis is that the researcher can leave a project when s/he has successfully established the innovation as self-sustaining, or if the adoption has failed and no clean-up work remains to be done. In successful cases, withdrawal from the project should be gradual rather than abrupt or it may endanger the success and cause harm to the project. Leaving a project after a failed introduction on the other hand obliges the researcher to clean up, say, revert changes to the code-base or reinstate previous infrastructure before a (gradual) withdrawal is in order.

7 How to Obtain Evaluation Results?

The actual evaluation of the invention under investigation is highly dependent on the nature of the invention itself and on the particular evaluation research goal. For some inventions the successful adoption itself can be a sufficient success, while others can only be judged by comparing product, process, or usage metrics to their baseline values prior to introduction. A third kind of invention might require the developers to be surveyed about their experience with the new technology.

Independent of these three basic approaches, the researcher will probably gain the most practical, albeit qualitative, insights for improving and assessing the invention by communicating with the project during the introduction. A researcher using the action research perspective may view this as the primary result.

8 Chances, Limitations and Conclusion

In the end, the question remains whether the experiences gained with introducing software engineering inventions in OSS projects can be applied to other settings (external validity). These might include differences in project sizes, application domains, software architectures, non-volunteer personnel, management, distribution and work-place setting, prior experience with software engineering methods, etc. The most common target setting is a revenue-dependent corporate environment. The following arguments argue why evaluation results from OSS projects may transfer to such environments: 1) Open Source developers are notorious for being critical of academic results, (2) availability of management championship and extrinsic motivations (like pay) can often spur adoption and use, and (3) full-time employees will benefit more from economies of scale and learning effects than part-time OSS developers.

The most notable limiting factor of our research approach is the restriction on the type of invention feasible for investigation. The diffusion of innovation literature lists several attributes of invention that will affect their rate of success for being introduced: (1) The *compatibility* of the invention with existing technology, values, and beliefs[2], (2) the intellectual and technical *complexity*, (3) the *observability* of the resulting effects of the invention, (4) the possibility to experiment with the invention (*trialability*) before committing to it, and (5) the *uncertainty* about the invention [22]. Halloran and Scherlis hypothesize more specifically with regards to OSS projects that these tend to distinguish sharply between trusted and untrusted contributions ("walled server" metaphor) and that inventions need to preserve this distinction to be applicable to OSS projects [15]).

This limits the approach as follow: while successful introduction suggests a valuable invention, failed introduction may be the result of specific properties of the OSS project (such as the walled-server) and may not say much about the real qualities of the invention.

As a second limitation we note that in contrast to fieldwork and ethnographic studies conducted with companies (see for instance [18]), it will be difficult to study the actual working processes and practices of each project participant since only the intermediates and process results, say, bug reports, CVS commits, and mailing list discussions are visible to the researcher. To gather information about the actual usage of tools on the computers of the project members, these need to be instrumented appropriately [23][16].

A third limitation of the approach concerns the speed at which adoption can occur. Open Source projects are to a large extent driven by volunteers who invest less than 10 hours per week and coordinate using asynchronous electronic means over different time zones [14]. The time scale of change should thus be expected to be much slower than in a commercial setting where employees work regular working hours and frequently interact synchronously.

Summing up, we have proposed to study the introduction of software engineering inventions to help researchers evaluate tools, methods, and processes developed in

[2] For instante, OSS projects may reject tools that are not licensed as Open Source software themselves.

academic settings, and have offered our preliminary results. While the research community can benefit from access to real life settings and the possibility to "feed back the community", the Open Source community is introduced to state-of-the-art inventions tailored to their specific problems by the inventors.

References

[1] W. B. Arthur. Increasing Returns and Path Dependence in the Economy. University of Michigan Press, 1994. ISBN: 0472064967.

[2] D. E. Avison, F. Lau, M. D. Myers, P. A. Nielsen. Action research. Commun. ACM, 42(1):94–97, 1999.

[3] M. Bakardjieva, A. Feenberg. Involving the virtual subject. Ethics and Information Technology, 2(4):233–240, 2001.

[4] J. Cassell. Ethical principles for conducting fieldwork. American Anthropologist, 82(1):28–41, March 1980.

[5] CMMI Product Team. Cmmi for development, version 1.2. Technical Report CMU/SEI-2006-TR-008, Software Engineering Institute, 2006.

[6] J. M. Corbin, A. Strauss. Grounded theory research: Procedures, canons, and evaluative criteria. Qualitative Sociology, 13(1):3–21, Mar. 1990.

[7] P. J. Denning, R. Dunham. Innovation as language action. Commun. ACM, 49(5):47–52, 2006.

[8] S. Dietze. *Modell und Optimierungsansatz für Open Source Softwareentwicklungsprozesse*. *Doktorarbeit*, Universität Potsdam, 2004.

[9] G.W. Downs, L. B. Mohr. Conceptual issues in study of innovation. Administrative Science Quarterly, 21(4):700–714, 1976.

[10] N. Ducheneaut. Socialization in an open source software community: A socio-technical analysis. Computer Supported Cooperative Work (CSCW), V14(4):323–368, Aug. 2005.

[11] M. L. Fennell. Synergy, influence, and information in the adoption of administrative innovations. Academy Of Management Journal, 27(1):113–129, 1984.

[12] N. Fligstein. Social skill and the theory of fields. Sociological Theory, 19(2):105–125, July 2001.

[13] M. S. Frankel, S. Siang. Ethical and legal aspects of human subjects research on the internet. Published by AAAS online , June 1999.

[14] R. A. Ghosh, B. Krieger, R. Glott, G. Robles, T. Wichmann. Free/Libre and Open Source Software: Survey and Study – FLOSS. Final Report, International Institute of Infonomics University of Maastricht, The Netherlands; Berlecon Research GmbH Berlin, Germany, June 2002.

[15] T. J. Halloran, W. L. Scherlis. High Quality and Open Source Software Practices. In J. Feller, B. Fitzgerald, F. Hecker, S. Hissam, K. Lakhani, and A. van der Hoek, editors, Meeting Challenges and Surviving Success: The 2nd Workshop on Open Source Software Engineering, pages 26– 28. ACM, 2002.

[16] P. M. Johnson, H. Kou, J. Agustin, C. Chan, C. Moore, J. Miglani, S. Zhen, W. E. J. Doane. Beyond the personal software process: metrics collection and analysis for the differently disciplined. In ICSE '03: Proceedings of the 25th International Conference on Software Engineering, pages 641–646, Washington, DC, USA, 2003. IEEE Computer Society.

[17] J. Law. Notes on the theory of the actor-network: Ordering, strategy and heterogeneity. Systems Practice, 5(4):379–393, 1992.

[18] T. C. Lethbridge, J. Singer. Experiences conducting studies of the work practices of software engineers. In H. Erdogmus and O. Tanir, editors, Advances in Software Engineering: Comprehension, Evaluation, and Evolution, pages 53–76. Springer, 2001.

[19] D. E. Perry, A. A. Porter, L. G. Votta. Empirical studies of software engineering: a roadmap. In Proceedings of the conference on The future of Software engineering, pages 345–355. ACM Press, 2000.

[20] L. Quintela García. *Die Kontaktaufnahme mit Open Source Software-Projekten. Eine Fallstudie.* Bachelor thesis, *Freie Universität* Berlin, 2006.

[21] E. S. Raymond. The Cathedral and the Bazaar. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1999. ISBN: 1565927249.

[22] E. M. Rogers. Diffusion of Innovations. Free Press, New York, 5th edition, August 2003. ISBN: 0743222091.

[23] F. Schlesinger, S. Jekutsch. ElectroCodeoGram: An environment for studying programming. In Workshop on Ethnographies of Code, Infolab21, Lancaster University, UK, March 2006.

[24] D. Stelzer, W. Mellis. Success factors of organizational change in software process improvement. Software Process: Improvement and Practice, 4(4):227–250, 1998.

[25] W. F. Tichy, P. Lukowicz, L. Prechelt, E. A. Heinz. Experimental evaluation in computer science: A quantitative study. Journal of Systems and Software, 28(1):9–18, Jan. 1995.

[26] G. von Krogh, S. Spaeth, K. Lakhani. Community, joining, and specialization in open source software innovation: A case study. Research Policy, 32:1217–1241(25), July 2003.

[27] R. K. Yin. Case Study Research: Design and Methods. Applied Social Research Methods. Sage Publications, Inc., 1988.

[28] M. V. Zelkowitz, D. R. Wallace. Experimental models for validating technology. Computer, 31(5):23–31, 1998.

3D Distributed Rendering and Optimization using Free Software

Carlos González-Morcillo, Gerhard Weiss, David Vallejo-Fernández, and Luis Jiménez-Linares, and Javier Albusac-Jiménez

The media industry is demanding high fidelity images for 3D synthesis projects. One of the main phases is Rendering, the process in which a 2D image can be obtained from the abstract definition of a 3D scene. Despite developing new techniques and algorithms, this process is computationally intensive and requires a lot of time to be done, especially when the source scene is complex or when photo-realistic images are required. This paper describes Yafrid (standing for Yeah! A Free Render grID) and MAgArRO (Multi Agent AppRoach to Rendering Optimization) architectures, which have been developed at the University of Castilla-La Mancha for distributed rendering optimization.

Keywords: Artificial Intelligence, Intelligent Agents, Optimization, Rendering.

1 Introduction

Physically based Rendering is the process of generating a 2D image from the abstract description of a 3D scene. The process of constructing a 2D image requires several phases including modelling, setting materials and textures, placing the virtual light sources, and rendering. Rendering algorithms take a definition of geometry, materials, textures, light sources, and virtual camera as input and produce an image (or a sequence of images in the case of animations) as output. High-quality photorealistic rendering of complex scenes is one of the key goals of computer graphics. Unfortunately, this process is computationally intensive and requires a lot of time to be done when the rendering technique simulates global illumination. Depending on the rendering method and the scene characteristics, the generation of a single high quality image may take several hours (or even days!). For this reason, the rendering phase is often considered as a bottleneck in photorealistic projects.

To solve this problem, several approaches based on parallel and distributed processing have been developed. One of the most popular is the render farm: a computer cluster owned by an organization in which each frame of an animation is independently calculated by a single processor. There are new approaches called Computational Grids which use the Internet to share CPU cycles. In this context, Yafrid is a computational Grid that distributes the rendering of a scene among a large number of heterogeneous computers connected to the Internet.

This paper describes the work flow and the free software tools used at the University of Castilla-La Mancha in several 3D rendering projects based on Open Source Cluster Ap-

Authors

Carlos Gonzalez-Morcillo is an assistant professor and a Ph.D. student in the ORETO research group at the University of Castilla-La Mancha. His recent research topics are multi-agent systems, distributed rendering, and fuzzy logic. He received both B.Sc. and M.Sc. degrees in Computer Science from the University of Castilla-La Mancha in 2002 and 2004 respectively. <carlos.gonzalez@uclm.es>.

Gerhard Weiss is the scientific director at SCCH (Software Competence Center Hagenberg GmbH), one of Austria's largest independent research centres. His main interests have been in computational intelligence and autonomous systems in general, and in the foundations and application of agent and multi-agent technology in particular. He is the co/editor of the reference book in this area "Multiagent Systems" (MIT Press). <gerhard.weiss@scch.at>.

David Vallejo-Fernandez is an assistant professor and a Ph.D. student in the ORETO research group at the University of Castilla-La Mancha. His recent research topics are multi-agent systems, cognitive surveillance architectures, and distributed rendering. He received his B.Sc. degree in Computer Science from the University of Castilla-La Mancha in 2006. <david.vallejo@uclm.es>

Luis Jimenez-Linares is an Associate Professor of Computer Science at the University of Castilla-La Mancha. His recent research topics are multi-agent systems, knowledge representation, ontology design, and fuzzy logic. He received both M.Sc. and Ph.D. degrees in Computer Science from the University of Granada in 1991 and 1997 respectively. He is member of the European Society of Fuzzy Logic and Technology. <luis.jimenez@uclm.es>.

Javier Albusac-Jimenez is a researcher and a Ph.D. student in the ORETO research group at the University of Castilla-La Mancha. His recent research topics are multi-agent systems, cognitive surveillance, and cognitive vision. He received his B.Sc. degree in Computer Science from the University of Castilla-La Mancha in 2005. <javieralonso.albusac@uclm.es>.

plication Resources (OSCAR) and Blender & Yafray render engines), as well as our new research software distributed under General Public Licence (GPL). Going into detail, the global architecture of Yafrid and the optimization system (based on principles from the area of multi-agent systems) called MAgArRO are exposed. This last system uses expert knowledge to make local optimizations in distributed rendering. Finally, some experimental results which show the benefits of using these distributed approaches are presented. The paper is structured as follows. The following section overviews the state of the art and the current main research lines in rendering optimization. Thereby, the focus is on the issues related to parallel and distributed rendering. The next sections describe the general architecture of an OSCAR-based cluster, the Grid-based rendering system called Yafrid and the Distributed Intelligent Optimization Architecture called MAgArRO. In the next section, empirical results that have been obtained by using these systems are shown. The final section is dedicated to a careful discussion and concluding remarks.

1.1 Related Work

There are a many rendering methods and algorithms, each having different characteristics and properties [11][6] [10]. However, as pointed out by Kajiya [6], all rendering algorithms aim to model the light behaviour over various types of surfaces and try to solve the so-called rendering equation which forms the mathematical basis of all rendering algorithms. Common to these algorithms, the different levels of realism are related to the complexity and the computational time required to be done. Chalmers et al. [3] expose various research lines in rendering optimization issues.

Optimizations via Hardware. One method to decrease time is to make special optimizations using hardware. In this research line there are different approaches; some methods use programmable GPUs (Graphics Processing Units) as massively parallel, powerful streaming processors which run specialised code portions of a raytracer. The use of programmable GPUs out-performs the standard workstation CPUs by over a factor of seven [2]. The use of the CPU in conjunction with the GPU requires new paradigms and alternatives to the traditional architectures. For example, the architectural configurations proposed by Rajagopalan et al. [8] demonstrate the use of a GPU to work on real-time rendering of complex data sets which demand complex computations. There are some render engines designed to be used with GPU acceleration, such as Parthenon Renderer [5], which use the floating-point of the GPU, or the Gelato render engine, which works with Nvidia graphic cards.

Optimizations using distributed computing. If we divide the problem into a number of smaller problems (each of them being solved on a separate processor), the time required to solve the full problem would be reduced. In spite of being true in general, there are many distributed rendering problems that would be solved. To obtain a good solution to a full problem on a distributed system, all processing elements must be fully utilized. Therefore, a good task scheduling strategy must be chosen. In a domain decomposition strategy [3], each processing unit has the same algorithm, and the problem domain is divided to be solved by the processors. The domain decomposition can be done using a data driven or a demand driven approach. In a data driven model, the tasks are assigned to the processing units before starting to compute. In the other alternative, the demand driven model, the tasks are dynamically allocated when the processing units become idle. This is done by implementing a pool of available tasks. This way, the processing units make a request for pending work.

In both models (data and demand driven), a cost estimation function of each task is needed. This cost prediction is very difficult to exactly calculate before completing the image due to the nature of global illumination algorithms (unpredictable ray interactions and random paths of light samples).

The biggest group of distributed and parallel rendering systems is formed by dedicated clusters and rendering farms used by some 3D animation companies. Depending on the task division, we can talk about fine-grained systems, in which each image is divided into small parts that are sent to a processor to be independently done, or coarse-grained (in case of animations) in which each frame of an animation is entirely done by one processing unit. In this context, Dr. Queue [17] is an open source tool designed for distributing frames through a farm of networked computers. This multiplatform software works in a coarse-grained division level. In Section 2, our solution based on OSCAR open cluster [18] is exposed.

New approaches of distributed rendering use a grid design to allocate the tasks among a large number of heterogeneous computers connected to the Internet, using the idle time of the processor [1]. This emerging technology is called Volunteer Computing or Peer-to-peer computing, and is currently used in some projects based on the BOINC technology (such as BURP [16] Big and Ugly Rendering Project). In Section 3, the main architecture of Yafrid and its key advantages are exposed.

Cost prediction. The knowledge about the cost distribution across the scene (i.e. across the different parts of a partitioned scene) can significantly aid the allocation of resources when using a distributed approach. This estimation is absolutely necessary in commercial rendering productions, to assure deadlines and provide accurate quotations. There are many approaches based on knowledge about cost distribution; a good example is [9]. In Section 4.1, the cost prediction mechanism used in MAgArRO is exposed.

Distributed Multi-Agent Optimization. The distribution of multi-agent systems and their properties of intelligent interaction allow us to get an alternative view of rendering optimization. The work presented by Rangel-Kuoppa [7] uses a JADE-based implementation of a multi-agent platform to distribute interactive rendering tasks on a network. Although this work employs the multi-agent metaphor, it does not make use of multi-agent technology itself. The MAgArRO architecture proposed in Section 4 is an ex-

ample of a free and Distributed Multi-Agent architecture which employs expert knowledge to optimize the rendering parameters.

2 OSCAR-based Cluster Approach

Nowadays, Universities have good practical classrooms provided with plenty of computers. This equipment is frequently maintained and updated. Nevertheless, these computers are inactive over vacation and at night. This existing hardware infrastructure can be co-ordinated during idle time by using free software thus creating clusters and low-cost supercomputers [14]. OSCAR [18] is a software platform which allows the user to deploy clusters based on GNU/Linux. In the next section, the general architecture of the OSCAR-based system will be explained. This tool is being used at the University of Castilla-La Mancha to render 3D projects [20][22].

2.1 Architectural Overview

In our execution environment, the system is composed of 130 heterogeneous workstations placed in different classrooms. Every classroom has a specific hardware type (based on x86 architecture). The minimal requirements to belong to the system are 500MB of RAM, a swap partition of 1GB, and a connection of at least 100Mbits/s (all computers are connected to one network using 100 Mbits/s switches). The Figure 1 illustrates these requirements.

The classrooms, where OSCAR cluster is used, are dedicated to education. For this reason, the best choice is not to permanently install any software in them. The subproject Thin-OSCAR [19] allows us to use machines without a local HD or a partition to install the operating system as members of the OSCAR cluster.

Each rendering node is configured obtaining the configuration parameters from the network. This is done by using the Pre eXecution Environment (PXE) extension of the BIOS. In our case, these data are the operating system image in which will be executed.

The server has two key processes to handle the PXE requests:

- DHCPD: the Dynamic Host Configuration Protocol daemon. This protocol is used to assign IP addresses to clients and to load the operating system image.
- TFTPD: the Trivial Transfer Protocol daemon. When the server receives a file request, it sends it to the client by using some configuration tables.

In order to begin and finish the execution of the computers in a controlled schedule, the WOL (*Wake On Lan*) functionality of modern computers is used. These BIOS extensions are used with the help of the motherboard and the software package Ether-Wake (developed by Donal Becker). When the package generated by Ether-Wake arrives, the computer boots and loads the operating system image.

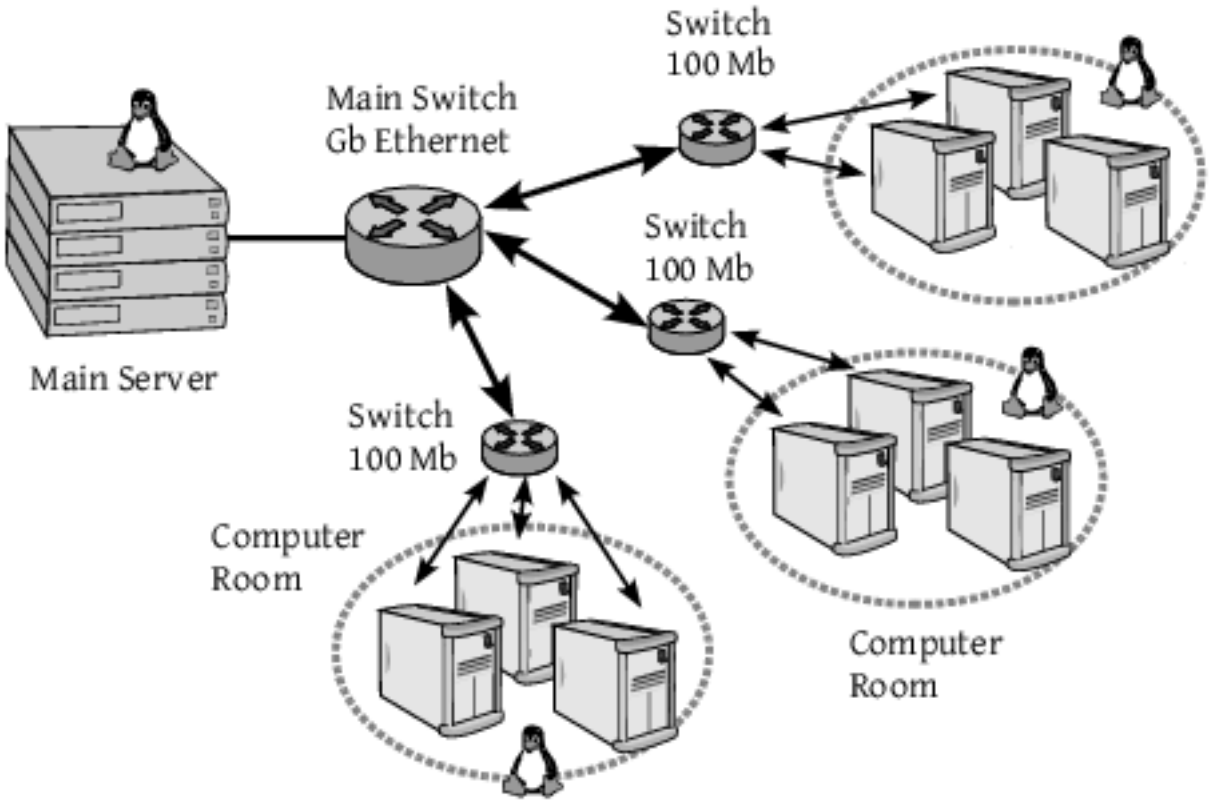


Figure 1: OSCAR-based Rendering Farm at ESI-UCLM.

has finished, the ACPI interface is used to halt them. The server establishes a ssh connection to each node and sends it a *shutdown* command.

3 Yafrid: Grid-based Rendering

Yafrid is basically a system which takes advantage of the characteristics of computational grids by distributing the rendering of a scene through the Internet. The system also has other important tasks related to the management of the workunits and the controlled use of the grid.

3.1 Architectural Overview

The top-level components of Yafrid are basically the following ones:

- **Server.** The hub of Yafrid. Its basic component is the Distributor which gets works from a queue and sends them to the providers.
- **Service Provider.** This entity processes the client requests.
- **Client.** A client is an external entity which does not belong to the system in a strict sense. Its role is to submit works to the providers. Those works are stored in a queue used by the distributor to take the next one to be scheduled.

In terms of access to the system, three user roles have been defined to determine the user access privileges:

- **Client.** With this role, a user is allowed to submit works to the grid. A client is also able to create and manage render groups (clients and providers can subscribe to these groups). When a project is created, it can belong to a group. In this case, only providers belonging to the same group can take part in the project rendering.
- **Administrator.** This role is needed for operating the whole system and has complete privileges to access to the information about all the users.
- **Provider.** The provider is a role user that has installed the software needed for receiving works. Providers can access to their own information and some statistics.

Yafrid server. The server is the fundamental node for setting the Yafrid render system up. Each one of the providers connects to this node in order to let the grid to use its CPU cycles for rendering the scenes submitted by Yafrid clients. Yafrid server consists of an architecture of four layers (Figure 2). This design is loosely based on the architecture that appears in [4]. Those layers are "*Resource Layer*", "*Service Layer*", "*Yafrid Server*", and "*User Layer*" (from lowest to highest level of abstraction).

Resource Layer. This layer has the lowest abstraction level and it is the most related with operating system issues. The resource layer has the following components:

- **Database system.** It is in this database where the tables needed for the correct operation of the system are maintained. Some of these tables are used to obtain statistics about the system performance, whereas other ones store the data associated to users, groups, projects, etc. The current implementation uses MySQL.

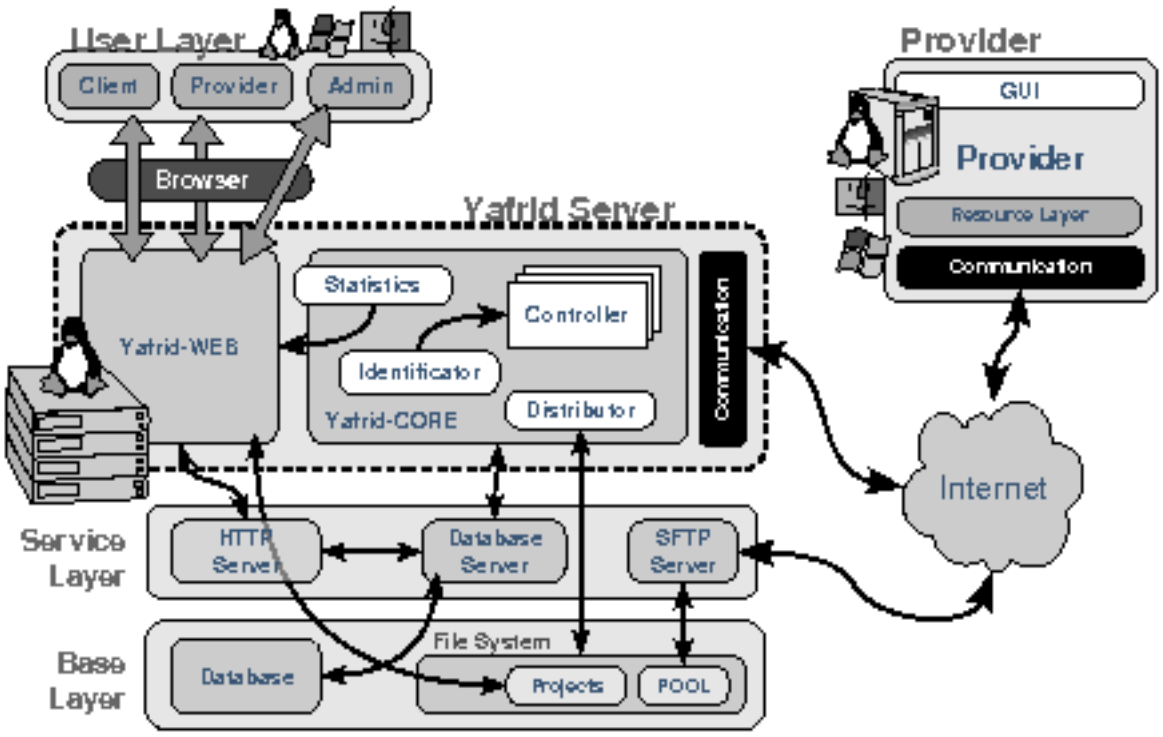


Figure 2: Yafrid General Architecture.

- **Filesystem.** Sometimes, it is necessary to directly access the file system from the high-level layers. Basically, the system distinguishes two types of directories. There are some directories which are used to store the workunits of projects that will be accessed via SFTP by providers. Those directories compose the workunit POOL. The other category of directories is composed by those directories that contain information about users and projects.
- **Network system.** The module dedicated to communications hides the use of network resources by using a middleware (the current implementation uses ZeroC ICE [25]).

Service Layer. Basically, this layer contains the different servers that allow modules to access resources that belong to lower layers. There are several servers at this level:

- **HTTP Server.** The Yafrid-WEB module is established over this server. As Yafrid-WEB has been developed using dynamic web pages written in a web-oriented scripting language (the current implementation uses PHP), the web server must support this language.
- **Database server.** This server is used by the different Yafrid modules to access to the indispensable data for the system operation.
- **SFTP server.** This server is accessed by the service providers to obtain the files needed for carrying out the rendering of the work units. Once the rendering has finished, the SFTP server will be used to send the resultant image to the Yafrid Server.

Yafrid Layer. This is the main layer of the server and it is composed of two different modules (Yafrid-WEB and Yafrid-CORE) working independently. **Yafrid-WEB** is the interactive module of the server and it has been developed as a set of dynamic web pages. **Yafrid-CORE** is the non-interactive part of the server. This module has been mainly developed using Python. Yafrid-CORE is composed of three submodules: Distributor, Identificator, and Statistics.

- The **Distributor** is the active part of the server. It implements the main algorithm in charge of doing the indispensable tasks, such as generating the work units, assigning them to providers, controlling the timeout, finishing projects, and composing the results. With the results generated by the different providers, the distributor composes the final image. This process is not trivial because slight differences between fragments obtained from different computers can be distinguished (due to the random component of Monte Carlo based methods as Path-tracing). For that reason, it is necessary to smooth the joint between fragments which are neighbours using a lineal interpolation mask. We define a zone in the work unit that is combined with other work units in the server. In Figure 3 on the left, we can see problems when joining the work units if we do not use a blending method.
- The passive part of Yafrid-CORE is called the **Identificator** module. Its mission consists of waiting for the communications from the providers. The first time a provider tries to connect to the Yafrid server, the Identificator generates an object (the provider controller) and returns a proxy to this object. Each provider has its own controller.
- **Provider.** The provider is the software used by the users who want to give CPU cycles to the grid. It can work in both visual and non-visual mode. First, the provider must connect to the grid. Once activated, the provider waits until the server sends a work unit to process. After finishing the rendering,

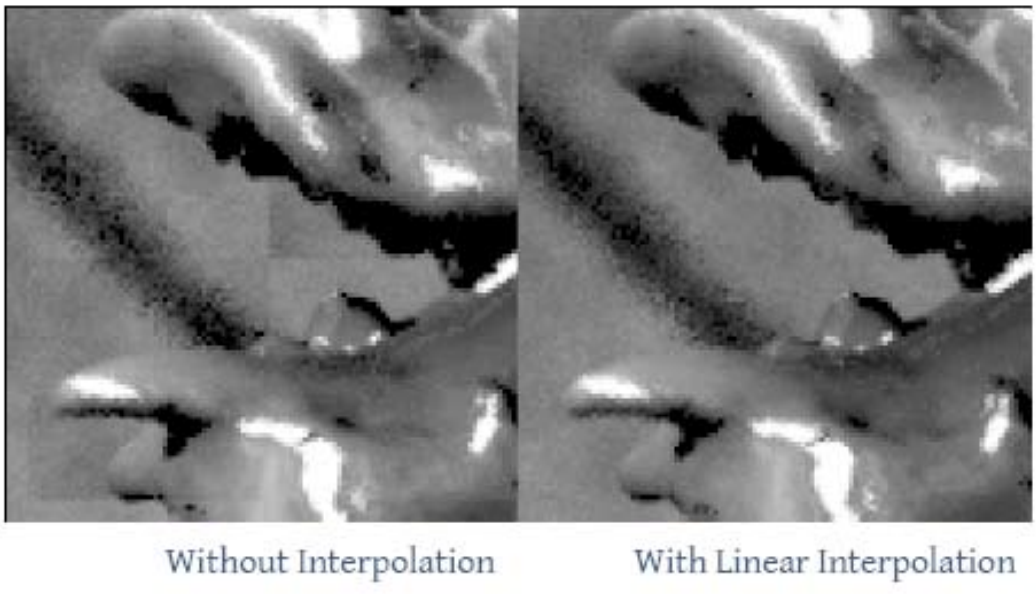


Figure 3: Artifacts without Interpolation between Workunits.

the provider sends the file via SFTP and informs the controller that the work was done.

4 MAgArRO: Distributed Intelligent Optimization

According to [12], an agent is a computer system that is situated in some environment and that is capable of actioning in this environment in order to meet its design objectives. MAgArRO uses the principles, techniques, and concepts known from the area of multi-agent systems, and it is based on the design principles of FIPA (Foundation for Intelligent Physical Agents) standards [21].

MAgArRO has also been developed using the ICE middleware [25]. The location service IceGrid is used to indicate in which computer the services reside. Glacier2 is used to solve the difficulties related with hostile network environments, being the agents able to connect behind a router or a firewall.

4.1 Architectural Overview

As mentioned, the overall architecture of MAgArRO is based on the design principles of FIPA standards. In Figure 4, the general workflow and the main architectural roles are shown. In addition to the basic FIPA services, MAgArRO includes specific services related to Rendering Optimization. Specifically, a service called Analyst studies the scene in order to enable the division of the rendering tasks. A blackboard is used to represent some aspects of the common environment of the agents. Finally, a master service called *Master* handles dynamic groups of agents who cooperate by fulfilling subtasks.

Figure 4 also illustrates the basic workflow in MAgArRO (the circled numbers in this figure represent the following steps).

1) The first step is the subscription of the agents to the system. This subscription can be done at any moment; the available agents are dynamically managed. When the system receives a new file to be rendered, it is delivered to the Analyst service.

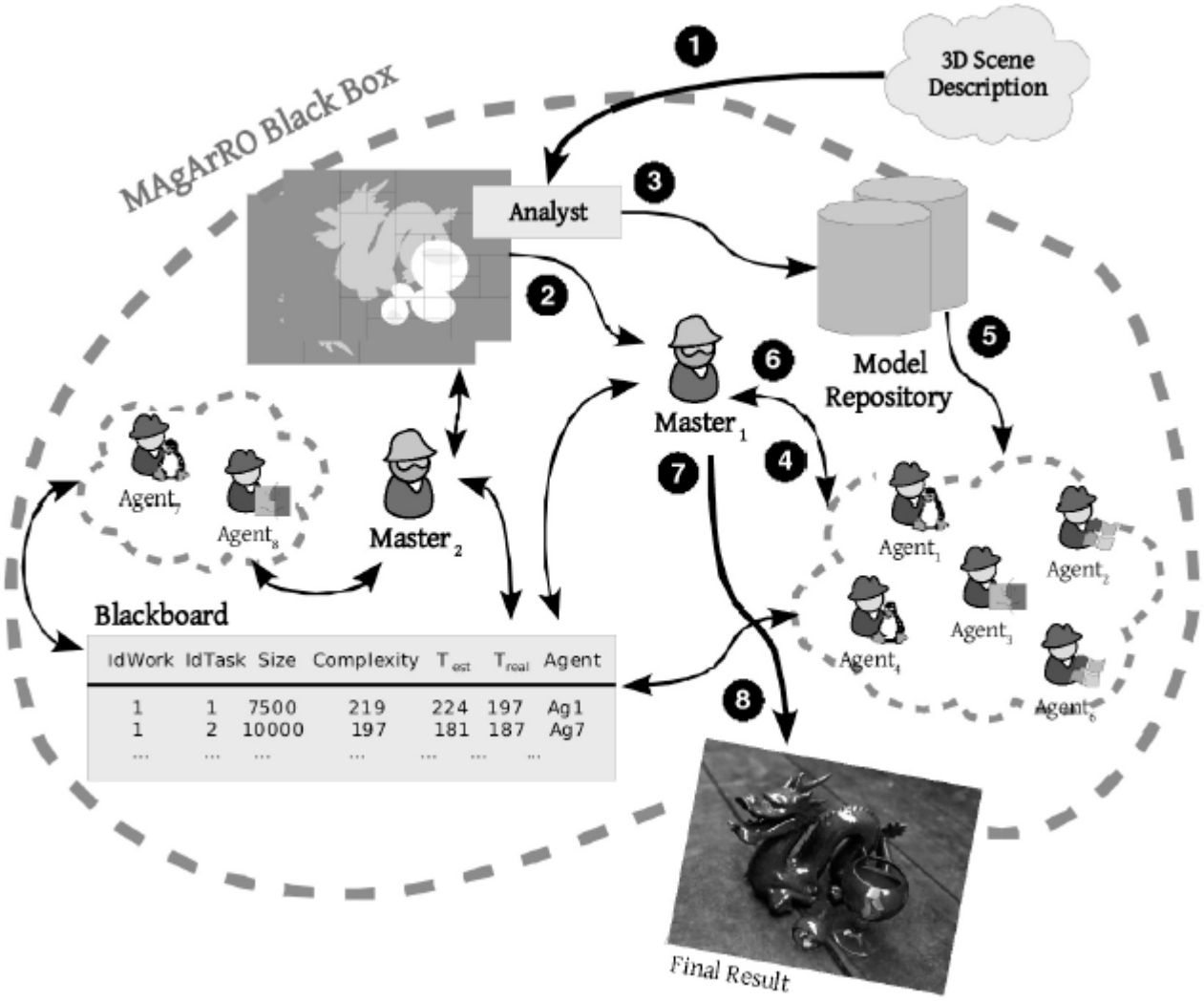


Figure 4: General Workflow and Main Architectural Roles.

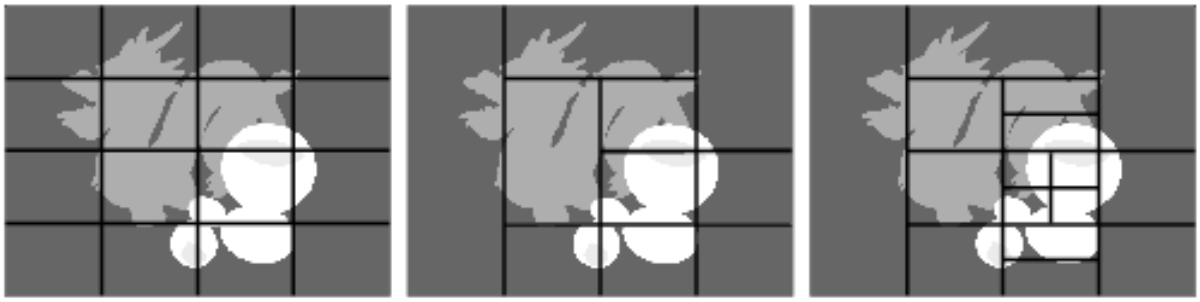


Figure 5: Importance Maps. Left: Blind Partitioning (First Level). Center: Join Zones with Similar Complexity (Second Level). Right: Balancing Complexity/Size Ratio (Third Level).

2) The Analyst analyzes the scene, making some partitions of the work and extracting a set of tasks.

3) The Master is notified about the new scene which is sent to the Model Repository.

4) Some of the agents available at this moment are managed by the Master and notified about the new scene.

5) Each agent obtains the 3D model from the repository and begins to auction.

6) The (sub-)tasks are executed by the agents and the results are sent to the Master.

7) The final result is composed by the Master using the output of the tasks previously done.

8) The Master sends the rendered image to the user. Key issues of this workflow are described in the following section.

Analysis of the Scene using Importance Maps. MAgArRO employs the idea of estimating the complexity of the different tasks in order to achieve load-balanced partitioning. Complexity analysis is done by the Analyst agent prior to (and independent of) all other rendering steps. The main objective in this partitioning process is to obtain tasks with similar complexity to avoid the delay in the final time caused by too complex tasks. This analysis may be done in a fast way independently of the final render process.

Once the importance map is generated, a partition is constructed to obtain a final set of tasks. These partitions are hierarchically formed at different levels, where at each level the partitioning results obtained at the previous level are used. At the first level, the partition is made taking care of the minimum size and the maximum complexity of each zone. With these two parameters, the *Analyst* makes a recursive division of the zones (see Figure 5). At the second level, neighbour zones with similar complexity are joined. Finally, at the third level the *Analyst* tries to obtain a balanced division where each zone has nearly the same complexity/size ratio. The idea behind this division is to obtain tasks that all require roughly the same rendering time. As shown below in the experimental results, the quality of this partitioning is highly correlated to the final rendering time.

Using Expert Knowledge. When a task is assigned to an agent, a set of fuzzy rules is used to model the expert knowledge and to optimize the rendering parameters for this task. Sets of fuzzy rule are considered well suited for expert knowledge modelling due to their descriptive power and easy extensibility [13]. The output parameters (i.e. the consequent part of the rules) are configured so that the time required to complete the rendering is reduced and the loss of quality is minimized. Each agent may model different expert knowledge with a different set of fuzzy rules. For example, the following rule is used (in a set of 28 rules) for describing the rendering parameters of the Pathtracing method: R_1: **If** C is {B,VB} and S is {B,N} and Op is VB **then** Ls is VS and Rl is VS.

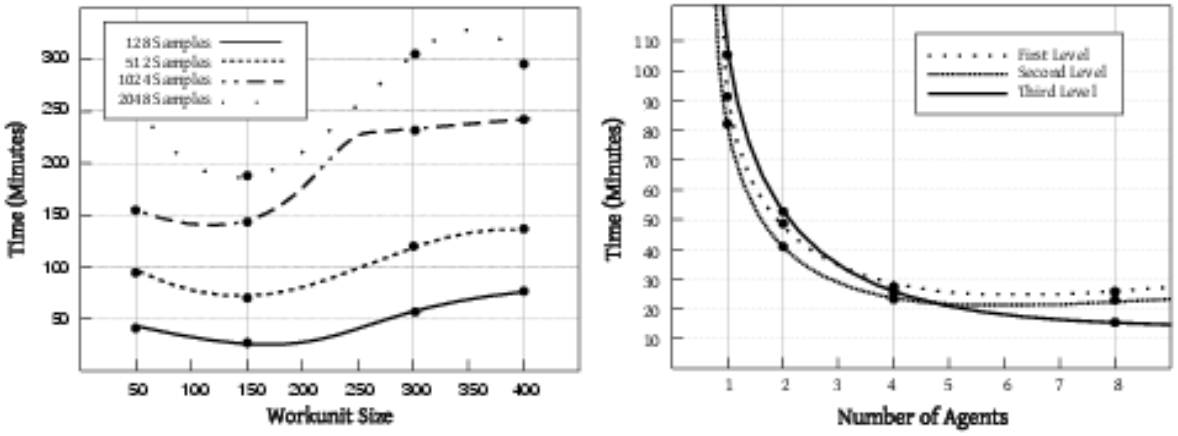


Figure 6: Left: Yafrid. Rendering Time Related to Workunit Size. Right: MAgArRO. Different Levels of Partitioning with a Normal Optimization Level.

The meaning of this rule is "If the Complexity is Big or Very Big and the Size is Big or Normal and Optimization Level is Very Big, then the number of Light Samples is Very Small and the Recursion Level is Very Small". The Complexity parameter represents the complexity/size ratio of the task, the Size represents the size of the task in pixels, and the Optimization Level is selected by the user. The output parameter Recursion Level defines the global recursion level in raytracing (number of light bounces), and the Light Samples defines the number of samples per light in the scene (higher values involve more quality and more rendering time).

5 Experimental Results

In order to test the behaviour of the systems, 8 computers with the same characteristics were connected to Yafrid and MAgArRO. These nodes (Intel Pentium Centrino 2 GHz, 1GB RAM) were used in both systems during the execution of all the tests. The test scene contained more than 100,000 faces, 5 levels of raytracing recursion in mirror surfaces (the dragon), 6 levels in transparent surfaces (the glass), 128 samples per light source, and was rendered using the free render engine Yafray [23]. In addition, 200,000 photons were released in order to construct the Photon Map structure. With this configuration, the rendering on a single machine without optimizations took 121:17 (121 minutes and 17 seconds).

In the case of Yafrid, as we can see in Figure 6 (*Left*), the rendering time in the best case is nearly seven times better using the grid, and less than twice as good in the worst case. With these results, it is clear the importance of choosing an appropriate workunit size. This occurs because there are complex tasks that slow down the whole rendering process even if the number of nodes is increased.

As we mentioned, MAgArRO uses *Importance Maps* to estimate the complexity of the different tasks. Figure 6 (*Right*) shows the time required by using different partitioning levels. Using a simple first-level partitioning (similar to the Yafrid approach), a good render time can be obtained with just a few agents. However, when the number of agents (processing nodes) grows, the overall performance of the system increases because the differences in the complexity of the tasks are relatively small.

As a final remark, note that intelligent optimization may result in different quality levels for different areas of the overall scene. This is because more aggressive optimization levels (Big or Very Big) may result in a loss of detail. For example, in Figure 7.e, the reflections on the glass are not as detailed as in Figure 7.a. The difference between the optimal render and the most aggressive optimization level (Figure 7.f) is minimal.

6 Discussion and Conclusion

The computational requirements of photo-realistic rendering are huge and, therefore, to obtain the results in a reasonable time and on a single computer is practically impossible (even more difficult in the case of animations). Several approaches based on different technologies have been exposed in this paper.

Our **OSCAR**-based cluster has some interesting characteristics:

- Very good throughput in the case of animations. The system divides each frame of the animation into different nodes of the cluster. The fine-grained approach needs the programming of new features in the main server.
- The processing nodes are used during idle time (at night).
- The latency due to the file transfer is minimal (thanks to the use of a Fast Ethernet network).

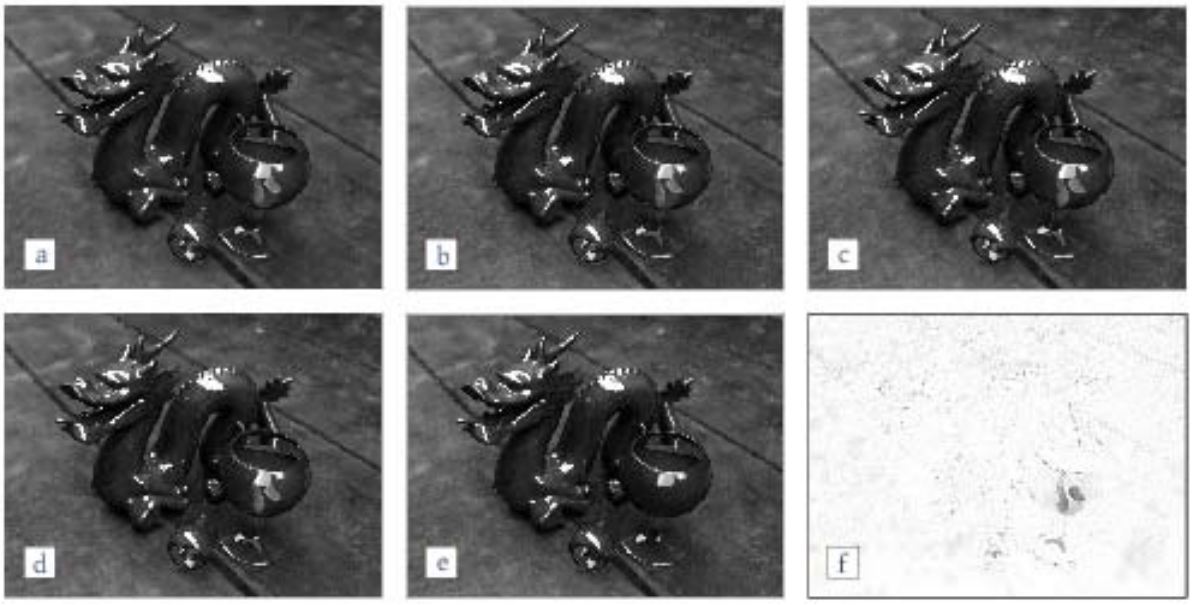


Figure 7: Result of the Rendering Using Different Optimization Levels. (a) No Optimization and Render in one Machine. (b) Very Small (c) Small (d) Normal (e) Very Big (f) Difference between (a) and (e) (the Lighter Colour, the Smaller Difference).

Otherwise, the cluster can only be used by submitting tasks to the main server into the same organization.To solve some of these problems, the **Yafrid** approach was designed. This computational grid has some important advantages:

- There is no cluster; the providers can be heterogeneous (software and hardware) and can be geographically distributed.
- With the fine-grained approach, we can make local optimizations in each frame.
- One of the main advantages of this distributed approach is the scalability. The performance perceived by the user depends on the number of subscribed providers.

Some enhancements should be done to improve the Yafrid performance. Some of them were added to **MagArRO**:

- MAgArRO enables importance-driven rendering through the use of importance maps.
- It allows us to use expert knowledge by employing flexible fuzzy rules.
- It applies the principles of decentralized control and local optimization. The services are easily replicable. Thus, possible bottlenecks in the final deployment can be minimized.

There are many future research lines. In our current work, we concentrate on the combination of the best characteristics of Yafrid and MAgArRO to integrate the new system (called YafridNG) in the official Blender branch [15]. The source code of these systems, distributed under GPL license, can be downloaded at [24].

Acknowledgments

This work has been funded by the Consejeria de Ciencia y Tecnología and the Junta de Comunidades de Castilla-La Mancha under Research Projects PAC-06-0141 and PBC06-0064. Special thanks to Javier Ayllon for his support at the Supercomputation Service (University of Castilla-La Mancha).

References

[1] D.P. Anderson, G. Fedak. The Computational and Storage Potential of Volunteer Computing. Sixth IEEE International Symposium on Cluster Computer and the Grid (CCGRID '06). p. 73-80. May 2006.

[2] I.Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, P. Hanrahan. Brook for GPUs: Stream Computing on Graphics Hardware. Proceedings of SIGGRAPH '04, p.777-786.

[3] A. Chalmers, T. Davis, E. Reinhard. Practical Parallel Rendering. Ed. A. K. Peters, 2002. ISBN: 1-56881-179-9.

[4] I. Foster, C. Kesselman, S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International Journal of Supercomputing Applications 15, 3(2002).

[5] T. Hachisuka. High-Quality Global Illumination Rendering using Rasterization. GPU Gems 2: Programming Techniques for High Performance Graphics and General-Purpose Computation. Addison-Wesley Professional, 2005.

[6] J.T. Kajiya. The rendering equation. Computer Graphics 20(4): 143-150. Proceedings of SIGGRAPH '86.

[7] R.R. Kuoppa, C.A. Cruz, D. Mould. Distributed 3D Rendering System in a Multi-Agent Platform. Proceedings of the Fourth Mexican International Conference on Computer Science, 8, 2003.

[8] R. Rajagopalan, D. Goswami, S.P. Mudur. Functionality Distribution for Parallel Rendering. Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05), p. 18-28, April 2005.

[9] E. Reinhard, A.J. Kok, F.W. Jansen. Cost Prediction in Ray Tracing. Rendering Techniques '96, p. 41-50. Springer-Verlag, June 1996.

[10] E. Veach, L.J. Guibas. Metropolis light transport. Proceedings of SIGGRAPH '97, p. 65-76. New York, USA: ACM Press - Addison Wesley Publishing Co.

[11] T. Whitted. An improved illumination model for shaded display. Proceedings of SIGGRAPH '79, 14. New York, USA: ACM Press.

[12] M.J. Wooldridge. An introduction to multiagent systems. John Wiley & Sons, 2002. ISBN: 0-471-49691-X

[13] L.A. Zadeh. The concept of a linguistic variable and its applications to approximate reasoning. Information Science, 1975.

[14] Beowulf: Open Scalable Performance Clusters. <http://www.beowulf.org>.

[15] Blender: Free 3D content creation suite. <http://www.blender.org>.

[16] BURP: Big Ugly Rendering Project. <http://burp.boinc.dk/>.

[17] Dr. Queue.: OS Software for Distributed Rendering. <http://www.drqueue.org/>.

[18] OSCAR: Open Cluster Group. <http://www.openclustergroup.org/>.

[19] Thin-OSCAR. <http://thin-osccar.sourceforge.net/>.

[20] Virtual Tour ESI UCLM. <http://www.inf-cr.uclm.es/virtual/index_en.html>.

[21] FIPA. Foundation for Intelligent Physical Agents. <http://www.fipa.org>.

[22] Virtual Visit - Hospital Ciudad Real. <http://dev.oreto.inf-cr.uclm.es/www/vvhosp>.

[23] Yafray: Yet Another Free Raytracer <http://www.yafray.org>.

[24] Yafrid Next Generation. <http://www.yafridng.org>.

[25] ZeroC ICE Middleware. <http://www.zeroc.com>.

Identifying Success and Tragedy of FLOSS Commons: A Preliminary Classification of Sourceforge.net Projects

Robert English and Charles M. Schweik

This article was previously published in the Proceedings of the First International Workshop on Emerging Trends in FLOSS Research and Development. FLOSS '07. ISBN: 0-7695-2961-5. Digital Object Identifier: 10.1109/FLOSS.2007.9. It is reproduced with kind permission of IEEE and the author.

Free/Libre and Open Source Software (FLOSS) projects are a form of commons where individuals work collectively to produce software that is a public, rather than a private, good. The famous phrase "Tragedy of the Commons" describes a situation where a natural resource commons, such as a pasture, or a water supply, gets depleted because of overuse. The tragedy in FLOSS commons is distinctly different: It occurs when collective action ceases before a software product is produced or reaches its full potential. This paper builds on previous work about defining success in FLOSS projects by taking a collective action perspective. We first report the results of interviews with FLOSS developers regarding our ideas about success and failure in FLOSS projects. Building on those interviews and previous work, we then describe our criteria for defining success/tragedy in FLOSS commons. Finally, we discuss the results of a preliminary classification of nearly all projects hosted on Sourceforge.net as of August 2006.

Keywords: Collective Action, FLOSS Commons, FLOSS Project, Project Abandonment, Project Classification, Project Failure, Project Success, Tragedy of the Commons.

1 Introduction

Free/Libre and Open Source Software projects (FLOSS) are recognized as Internet-based commons [1][13][15]. Since 1968, when the famous article "Tragedy of the Commons" by Garrett Hardin was published in the journal Science, there has been significant interest in understanding how to manage commons appropriately. Hardin's work, and much of the work that followed, focused on commons management in the natural environment. And in these commons, the "tragedy" Hardin described was over-harvesting and destruction of the resource, whether it be water, fish stock, forests, or our atmosphere. In FLOSS commons the "tragedy" is different; what developers hope to avoid is project abandonment and a "dead" project. In order for FLOSS projects to avoid tragedy and be successful, the collective action involved (or attempts at collective action in the case of projects with one participant) must be sustained at least until a software product has been produced. Discovering how FLOSS projects sustain collective action to produce useful software may have important implications for improving our understanding of FLOSS software development as well as computer-mediated collective action more generally [14][15].

In recent years, scholars have investigated different approaches to measuring the success and failure of FLOSS projects. For example, studies [2][3][7][11][16] measured FLOSS project "life" or "death" by monitoring project activity measures such as: (1) the release trajectory (e.g., movement from alpha to beta to stable release); (2) changes in version number; (3) changes in lines of

Authors

Robert C. English is a Research Fellow with the National Center for Digital Government at the University of Massachusetts Amherst and currently works with coauthor Charles M. Schweik on a major study on Free/Libre and Open Source Software (FLOSS) collaboration. He received his undergraduate degree in Information Management from UMass Amherst, and is currently a graduate student in Public Policy and Administration at the same institution. English had a successful business career before taking up his academic work. His research interests focus on potential uses of FLOSS and FLOSS collaboration methods in organizations working for social and political change. <renglish@pubpol.umass.edu>.

Charles M. Schweik is an Associate Professor in the Dept. of Natural Resources Conservation and the Center for Public Policy and Administration at the University of Massachusetts, Amherst, USA. He also is the Associate Director of the National Center for Digital Government at UMass Amherst <http://www.ncdg.org>. He has a PhD in Public Policy from Indiana University, a Masters in Public Administration from Syracuse University, and has an undergraduate degree in Computer Science. A primary research interest is in the use and management of public information technology. For more than six years, between his undergraduate degree and his MPA, he was a programmer with IBM. <cschweik@pubpol.umass.edu>.

code; (4) the number of "commits" or check-ins to a central storage repository, and (5) activity or vitality scores measured on collaborative platforms such as SF and Freshmeat.net. Weiss assessed project popularity using web search engines [17]. And most recently, Crowston, Howison and Annabi reviewed traditional models used to measure information systems success and then adapted them to FLOSS [4]. They collected data from Sourceforge.net (SF) and measured community size, bug-fixing time and the popularity of projects.

In this paper, we are trying to build on these studies by defining success and tragedy of FLOSS commons from the perspective of successful collective action. The section that follows this one describes interviews we conducted with FLOSS developers to get feedback on our ideas about defining success. Next, in the "Success and Tragedy Classification System" section, we define a 6-stage classification system of success and tragedy of FLOSS commons based on information gained from these interviews, as well as previous literature and our own earlier work studying FLOSS. In the "Operationalizing the Classification System" section, we describe our efforts in building a dataset which combines much of the August 2006 data available from the FLOSSmole project (described below) and data we gathered ourselves through automated data mining of the SF website. This section then describes how we operationalized our proposed success/tragedy classes using this dataset. The "Results" section discusses our preliminary classification of nearly all projects hosted on SF as of August 2006. We conclude the paper with some next steps.

2 FLOSS Developer Opinions on Success and Failure

We conducted eight interviews [18] with FLOSS developers between January and May of 2006 to get opinions about the independent variables we thought important to FLOSS project success and to get their thoughts about our definitions of success and tragedy. Because we wanted input from a diversity of projects, we stratified our sampling by the number of developers in the project. We created categories of projects with <5, 5-10, 11-25 and >25 developers and interviewed developers from two projects in each category. Interviews were conducted over the phone, digitally recorded, transcribed and analyzed using Transana 2 <http://www.transana.org>.

Interviews consisted of about sixty questions and took approximately one hour. Among other things, we asked interviewees how they would define success in a FLOSS project. Interviewees responded with five distinct views. One defined success in terms of the vibrancy of the project's developer community. Three defined FLOSS success as widely used software. Two others defined success as creating value for users. One developer cited achieving personal goals, and the last interviewee felt his project was successful because it created technology that percolated through other projects even though his project never produced a useful standalone product.

Immediately after asking interviewees about success, we asked how they would define failure in a FLOSS project. Interestingly, all eight developers said that failure had to do with a lack of users and two indicated that a lack of users leads to project abandonment. In a probing question that followed, we asked if defining a failed project as one that was abandoned before producing a release seemed reasonable. Four interviewees flatly agreed, three agreed with reservations and one disagreed. Two of those with reservations raised concerns about the quality of the release. For example, one project might not make its first release until it had a very stable, well functioning application while another project might release something that was nearly useless. Another interviewee had concerns about how much time could pass before a project was declared abandoned. One developer argued that a project that was abandoned before producing a release could be successful from the developer's point of view if he had improved his programming skills by participating. The dissenting developer felt that project source code would often be incorporated into other FLOSS projects and would not be a failure even if no release had been made.

So, how do these responses inform working definitions of success and tragedy? Because we view FLOSS projects as efforts in collective action with the goal of producing public good software, defining success in terms of producing a useful software product makes sense, and our interviewees seem to agree. Six of the eight interviewees suggested that success involves producing something useful for users. Since the real tragedy for a FLOSS project involves a failure to sustain collective action to produce, maintain or improve the software, defining failure in terms of project abandonment makes sense, and generally, our interviewees agreed. Treating the first release as a milestone or transition point between what we refer to as the "Initiation Stage" and the project "Growth Stage" [13][18] emerges logically from this line of thinking. All in all, these interviews supported our initial thinking about project success and tragedy.

3 A Success/Tragedy Classification System for FLOSS Commons

After conducting the interviews and considering the results, we developed a six-class system for describing success and tragedy of FLOSS projects across two longitudinal stages of Initiation and Growth (Table 1). In previous work [13][18] we defined "Initiation" as the start of the project to its first public release, and "Growth" as the period after this release [13, 18].

Therefore, a project is classified as (1) Success in the Initiation Stage (SI) when it has produced "a first public release". This can be easily measured for projects hosted at SF because SF lists all a project's releases. A project that is successful in the initiation phase automatically be-

comes an indeterminate project in the growth phase.

Projects are classified as (2) Tragedy in the Initiation Stage (TI) when the project is abandoned before producing a first public release. We define abandonment as few forum posts, few emails to email lists, no code commits or few other signs of project activity over a one-year period. Preliminary data we have analyzed from SF indicates that projects in Initiation that have not had a release for a year are generally abandoned (see the discussion of the "test sample" below).

A project is considered a (3) Success in the Growth Stage (SG) when it exhibits "three releases of a software product that performs a useful computing task for at least a few users (it has to be downloaded and used)". We decided that the time between the first release and the last release must be at least six months because a "growth stage" implies a meaningful time span. As mentioned above, we can easily measure the number of releases and the time between them since SF tracks this information. Measuring "a useful computing task" is harder and clearly more subjective. Acquiring the number of downloads recorded on project websites is probably the easiest measure, with the assumption that many downloads captures the concept of utility.

A project is considered a (4) Tragedy in the Growth Stage (TG) when it appears to be abandoned without having produced three releases or when it produced three releases but failed to produce a useful software product.

We classify a project as (5) Indeterminate in the Initiation Stage (II) when it has yet to reveal a first public release but shows significant developer activity.

Finally, projects are assigned (6) Indeterminate in the Growth Stage (IG) when they have not produced three releases but show development activity or when they have produced three releases over less than six months.

4 Operationalizing the Classification System

As a first step in operationalizing our definitions for FLOSS success and tragedy, we conducted a random test sample of sixty projects hosted on SF using April 2005 FLOSSmole data [5]. The FLOSSmole project is itself an open source-like project where researchers and others collaborate to collect and analyze data about FLOSS. The data is collected by automated "crawling" or "spidering" of SF and other open source hosting sites. We decided to conduct this test sample from the FLOSSmole database to look for problems with our classification scheme and to get some idea about the number of projects likely to fall within each of the classes. Following the logic used in our FLOSS developer interviews and knowing we wanted to study projects with larger numbers of developers because of their more interesting collective action issues, we stratified by number of developers into categories of <10, 10-25 and >25 developers. We randomly sampled twenty projects from each category for a total of sixty projects.

We chose 20 projects because it was a reasonable undertaking given time constraints and because twenty projects per category provided a standard error of plus or minus 22% with 95% probability for a binomial distribution. (Note: Because a project is either successful or failed, and either in the Initiation or Growth stage, the sample is a binomial distribution for these categories.) For these sixty sampled projects, we manually compiled data on project registration, last release date, number of downloads, project website URL and forum/email/postings among other data. From this data, we made a judgment about whether the software was "useful" and whether the project was abandoned. We classified the projects as SI, TI, SG or TG (see code definitions in the previous section) based on this information. We found no indeterminate cases in this sample.

Perhaps the most important information we acquired from the test sample is that the vast majority of projects that have not had a release for a year are abandoned. All 27 projects in the sample that (1) had not provided a release in over a year and (2) had less than three releases were abandoned. This finding suggested that we could produce a relatively simple but approximately accurate classification by using a project's failure to release within a year as a proxy for abandonment.

Naturally, operationalizing the definitions for success and tragedy measures had much to do with the availability of data. We chose to use the August 2006 data spidered from SF because it was the latest data available at the time we did our classification. This data has a total of 119,590 projects, but 235 of these projects are missing essential data leaving 119,355 projects. Although FLOSSmole had much of the data we needed for operationalizing our classification scheme, the data on the number of releases and the dates of the first and last release were not available. Consequently, we spidered that data ourselves between September 24, 2006 and October 16, 2006. Of the 119,355 projects, 8,422 projects had missing data or had been deleted from SF (SF occasionally purges defunct projects) between the August 2006 and the time we collected our data. The result: we have valid data for 110,933 projects. Based on our definitions described earlier, and the added information we gained from the test sample, we undertook a preliminary classification of SF projects as described in Table 1.

5 Results

Table 2 provides the number of SF projects classified by the two longitudinal stages: Initation and Growth. It also reports projects that could not be classified. Table 3 summarizes our results of our preliminary success and tragedy classification of all projects on SF and potential sources of error in our classifications.

We believe that the classification above is informative despite the possibility of classification errors (listed

Class/ Abbreviation	*Definition(D)/Operationalization(O)*
Success, Initiation (SI)	D: Developers have produced a first release. O: At least 1 release (Note: all projects in the growth stage are SI)
Tragedy, Initiation (TI)	D: Developers have not produced a first release and the project is abandoned O: 0 releases AND >=1 year since SF project registration
Success, Growth (SG)	D: Project has achieved three meaningful releases of the software and the software is deemed useful for at least a few users. O: 3 releases AND >= 6 months between releases AND does not meet the download criteria for tragedy detailed in the TG description below.
Tragedy, Growth (TG)	D: Project appears to be abandoned before producing 3 releases of a useful product. O: 1 or 2 releases and >=1 year since the last release at the time of data collection OR < 11 downloads during a time period greater than 6 months starting from the date of the first release and ending at the data collection date
Indeterminate Initiation (II)	D: Project has yet to reveal a first public release but shows significant developer activity O: 0 releases and < 1 year since project registration
Indeterminate Growth (IG)	D: Project has not yet produced three releases but shows development activity or has produced 3 releases but has been in the growth phase for less than 6 months. O: 1 or 2 releases and < 1 year since the last release OR 3 releases and < 6 months between releases

Table 1: Six FLOSS Success/Tragedy Classes and their Methods of Operazionalization.

in the third column of Table 3). Potential classification errors stem primarily from two sources: Source 1 Error- using one year without a release as a proxy for abandonment. Source 2 Error - using the number of downloads per month as a proxy for the software being useful.

Regarding Source 1 Error, our test sampling indicated with 95% probability that at most 22% of projects with less than 3 releases will turn out not to have had a release within a year and yet not be abandoned thus suggesting an upper bound for many abandonment errors.

As for Source 2 Error, some projects classified as TG may be useful and have met the download criteria for

tragedy or, on the other hand, some projects classified as SG may be useless and have not met download criteria for tragedy. Because our definition of SG is broad (the software performs a useful computing task for some number of users), we don't expect this error to be large. In other words, we expect that the vast majority of SG projects have produced something useful. Only 62 projects were classified as TG because they met the download criteria for Growth Stage tragedy in Table 1.

In terms of improving our classification, abandonment could be more precisely measured by (1) no code "commits" or changes in lines of code in the concurrent versioning system (CVS) or other repository over the course of a year, or (2) little or no activity on developer email lists and forums over the course of a year. Measures to improve our estimation of whether the software is useful could include: (1) a content analysis on utility of the software on data collected from user forums, e-mail archives or even web searches; (2) more carefully constructed download criteria that takes the life of the project and the availability of download data for different time periods into consideration. In addition, some projects make more

Stage	*# of Projects (% of Total classified)*
Initiation Stage	50,662 (47)
Growth Stage	57,085 (53)
Not classified	3,186*
Total classified	107,747
* These are valid projects, but could not be classified because they have 0 releases & downloads on SF but have other websites that may be used for these functions.	

Table 2: Sourceforge.net Projects Organized by Longitudinal Stage (as of August 2006).

Class	*# of Projects (%of Total)*	*Possible Classification Errors (other than errors in the SF data)*
TI	35,589 (33)	The project is not abandoned but > 1 year old
SG	15,782 (15)	The software is not used in spite of not meeting the download criteria for tragedy
TG	23,134 (21)	The project is not abandoned; OR The project produced useful software even though it met the download critera for tragedy
II	15,073 (14)	No classification errors (by definition)
IG	18,169 (17)	No classification errors (by definition)
Total	107,747	
Note: SI is not listed because these successes are Growth Stage projects. Including SI would double count.		

Table 3: Preliminary Classification of all FLOSS Projects on Sourceforge.net (as of August 2006).

than one release on a single day, thus bringing the criteria for three releases into question. We have data that will allow us to examine the time between each release and possibly refine the definition of the three release criteria, but this is yet to be done. Moreover, projects with web-sites not hosted on SF and no file releases or downloads on SF are currently not classified.

We hope to address these issues in future work.

6 Conclusion

Our most immediate task now is to validate the classification described above. We plan to sample a large enough number of projects to empirically establish the accuracy of our classification within a few percent. Our long-term goal is to use this classification as a dependent variable for quantitative models that investigate factors that lead to success and tragedy in FLOSS in the two stages of Initiation and Growth. We expect influential factors to be different in these two stages [13][18].

Despite the shortcomings of this classification system described in Section 5, we chose to publish preliminary results of our efforts in the spirit of "release early and often" and because defining and classifying success in FLOSS projects is so important to many FLOSS research projects. In the near future, we plan to release the data we collected and our classifications on the FLOSSmole site. We hope that in the tradition of open source collaboration other researchers will build on this work by correcting any perceived "bugs" in our approach and collecting additional data to improve classification accuracy.

Acknowledgments

Support for this study was provided by a grant from the U.S. National Science Foundation (NSFIIS 0447623). However, the findings, recommendations, and opinions expressed are those of the authors and do not necessarily reflect the views of the funding agency. Thanks go to Megan Conklin, Kevin Crowston and the FLOSSmole project team <http://ossmole .sourceforge.net/> for making their Sourceforge data available, and for their assistance. We are also grateful to Thomas Folz-Donahue for programming work building our FLOSS project database.

References

[1] D. Bollier. Silent Theft: The Private Plunder of Our Common Wealth, Routledge, London, 2002.

[2] A. Capiluppi, P. Lago, M. Morisio. "Evidences in the Evolution of OS projects through Changelog Analyses," In J.Feller, B. Fitzgerald, S. Hissam, and K. Lakhani (eds.) Taking Stock of the Bazaar: Proceedings of the 3rd Workshop on Open Source Software Engineering, 12 Dec. 2006. <http://opensource.ucc.ie/ icse2003>.

[3] K. Crowston, H. Annabi, J. Howison. "Defining Open Source Project Success," In Proceedings of the 24th Intl. Conference on Information Systems, ICIS, Seattle, 2003.

[4] K. Crowston, J. Howison, H. Annabi. "Information Systems Success In Free And Open Source Software Development: Theory And Measures," Software Process Improvement and Practice, v 11, n 2, March/ April, 2006, pp. 123-148.

[5] FLOSSmole. "sfProjectInfo06-Apr-2005," 16 June 2005;<http://sourceforge.net/project/showfiles.php?group_id=119453 &package_id=132043/>.

[7] S. Hissam, C. B. Weinstock, D. Plaksoh, J. Asundi. Perspectives on Open Source Software. Technical report CMU/SEI-2001-TR-019, Carnegie Mellon University. 10 Jan. 2007, <http://www.sei.cmu.edu/publications/documents/01.reports/01tr019.html>.

[11] M. Robles, G. Gonzalez, J.M. Barahona, J. Centeno-Gonzalez, V. Matellan-Olivera, L. Rodero-Merino. "Studying the Evolution of Libre Software Projects Using Publically Available Data," In J. Feller, B. Fitzgerald, S. Hissam, and K. Lakhani (eds.) Taking Stock of the Bazaar: Proceedings of the 3rd Workshop on Open Source Software Engineering, 12 Dec. 2006. <http://opensource.ucc.ie/icse2003>.

[13] C. Schweik. "An Institutional Analysis Approach to Studying Libre Software 'Commons'", Upgrade: The European Journal for the Informatics Professional, 10 Jan. 2007, <http://www.upgrade-cepis.org/issues/2005/3/up6-3Schweik.pdf>.

[14] C. Schweik, T. Evans, J. Grove."Open Source and Open Content: A Framework for Global Collaboration," in Social-Ecological Research. Ecology and Society 10 (1): 33. 10 Jan. 2007, <http://www.ecologyandsociety.org/vol10/iss1/ art33/>.

[15] C. Schweik. "Free / Open Source Software as a Framework for Scientific Collaboration," In Hess, Charlotte, and Elinor Ostrom, eds. Understanding Knowledge as a Commons: From Theory to Practice, MIT Press, Cambridge, Mass, 2007.

[16] K. J. Stewart, T. Ammeter. "An Exploratory Study of Factors Influencing the Level of Vitality and Popularity of Open Source Projects". In L. Applegate, R. Galliers, and J.I. DeGross (eds.) Proceedings of the 23rd International Conference on Information Systems, Barcelona, 2002, pp. 853-57.

[17] D. Weiss. "Measuring Success of Open Source Projects Using Web Search Engines", Proceedings of the First International Conference on Open Source Systems, Genova, 11th-15th July 2005. Marco Scotto and Giancarlo Succi (Eds.), Genoa, 2005, pp. 93-99.

[18] C. Schweik, R. English. "Tragedy of the FLOSS Commons? Investigating the Institutional Designs of Free/Libre and Open Source Software Projects", FirstMonday. 28 Feb. 2007, <http://www.firstmonday.org/issues/issue12_2/schweik/>.

CEPIS UPENET
UPGRADE European NETwork
The network of CEPIS member societies' publications

ICT Security

Security of Electronic Passports

Václav Matyáš, Zdeněk Říha, and Petr Švéda

This paper will be published, in Spanish, by ***Novática***. ***Novática***, <http://www.ati.es/novatica>, a founding member of **UP**ENET, is a bimonthly journal published, in Spanish, by the Spanish CEPIS society ATI (*Asociación de Técnicos de Informática* – Association of Computer Professionals).

The electronic part of the passport should increase the security of the whole document but at the same time brings in new threats to the privacy of the passport holder. Therefore electronic passports need to implement a new set of security features. This article discusses the principles and the effectiveness of these security features.

Keywords: Authentication, Basic Access Control, Electronic Passport, Entropy, Extended Access Control.

1 Introduction

A number of countries have already been issuing electronic passports for some time. The introduction of electronic passports has led to some controversial discussions. In this article we will be taking a look at some of the security features of electronic passports.

Passport features are specified by the International Civil Aviation Organization (ICAO), a UN agency, in its Document 9303. The sixth edition of Doc 9303 also introduces electronic passports [4]. Although the electronic part of the passport is still optional at a worldwide level, the US has asked all its Visa Waiver Program partners to introduce electronic passports and the European Union has agreed on the mandatory introduction of electronic passports in EU member states (to be more precise, this decision is not binding for the UK and Ireland, while three non-EU countries – Norway, Switzerland and Iceland – have opted in to the program).

The difference between a traditional passport and an electronic passport (ePassport) is that the latter has an embedded chip with a contactless interface (and the electronic passport logo on the front cover). The chip and the antenna are embedded in the cover or a page of the passport (see Figure 1). The chip is a contactless smart card compliant with ISO 14443 (either variant – A or B – is allowed). ISO 14443 based technology is designed to communicate over a distance of 0-10 cm and also supports relatively complex cryptographic chips and a permanent

Authors

Václav (Vashek) Matyáš is an Associate Professor at the Masaryk University Brno, Czech Republic, chairing its Department of Computer Systems and Communications. His research interests relate to applied cryptography and security, on which subject he has published over sixty peer-reviewed publications, including two books. He has worked with Microsoft Research Cambridge, University College Dublin, Ubilab at UBS AG, and was a Royal Society Postdoctoral Fellow with the Cambridge University Computer Lab. Dr. Vashek has edited the Computer and Communications Security Reviews, and has worked on the development of Common Criteria and with ISO/IEC JTC1 SC27. <matyas@fi.muni.cz>

Zdeněk Říha graduated from the Faculty of Informatics, Masaryk University in Brno, where he also received his Ph.D. in 2002. He works as an Assistant Professor at the Faculty of Informatics of the Masaryk University in Brno. He is currently seconded to the Joint Research Centre (JRC) of the European Commission at Ispra in Italy. His research interests include PKI, biometric systems, and the security of electronic documents and operating systems. <zriha@math.muni.cz>

Petr Švéda completed his Master degree in computer science, specializing in IT security and smart cards, in 2004 at the Masaryk University in Brno, CZ, and is now a PGS student there. Since 2004 he has been a researcher in the field of cryptographic protocols for restricted environments such as smart cards, mobile devices, and Wireless Sensor Networks at Masaryk University with an emphasis on the security of practical implementations. He has participated in development projects for academic, governmental, and commercial institutions in the Czech Republic. <xsveda@fi.muni.cz>

memory of so many kilobytes or megabytes. Here it differs from many other RFID technologies that are capable of communicating over longer distances but do not support operations more complicated than sending a simple identification bit string. The higher communication layer is based on classical smart card protocol ISO 7816-4 (i.e., commands like SELECT AID, SELECT FILE and READ BINARY are used).

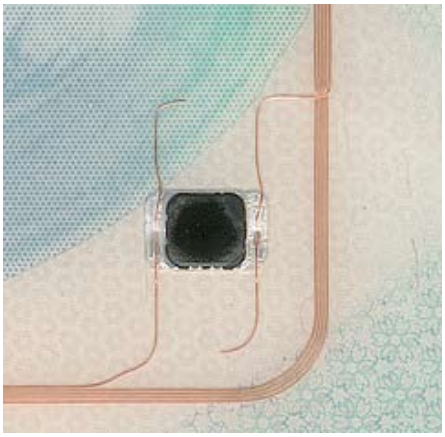


Figure 1: Contactless Chip and Antenna from British Passports.

The data in electronic passports is stored as files (elementary files in smart card terminology) in a single folder (dedicated file). Up to 16 data files named DG1 to DG16 (DG for Data Group) can hold the data. DG1 contains the data from the machine-readable zone (i.e., nationality, first name, surname, passport number, issuing state, sex, birth date, validity date, and optional data – for example a personal number), DG2 contains the photo of the passport holder (in JPEG or JPEG2000 and some additional metadata). DG3 is used for storing fingerprints, while DG4 may contain iris image data. The remaining data groups contain information about the holder, issuing institution, or the passport itself. Two additional files with metadata are also present. The file EF.COM contains a list of available data groups (and the information about versions used) and the file EF.SOD contains the digital signature of the data. Files EF.COM, EF.SOD, DG1 and DG2 are mandatory for all electronic passports. The data group DG3 will be mandatory in EU countries as from June 28, 2009 (and will be protected by an additional mechanism). All other data groups are optional.

2 Data Integrity (Passive Authentication)

The integrity of the stored information is protected by a digital signature available in the EF.SOD file. The file uses the SignedData structure of the CMS (Cryptographic Message Syntax) standard. The PKI hierarchy has a single level. Each country establishes its own CSCA (Country Signing CA), which certifies the authorities responsible for issuing the passports (e.g., state printers, embassies etc.). These authorities are called Document Signers. Data in the passport is then signed by one of these Document Signers.

To verify signatures, the CSCA certificates of the issuing country must be available and their integrity must be guaranteed (actually not such a simple task). The certificate of the Document Signer is either directly stored in the passport (in the certificate part of the SignedData structure – this is mandatory in the EU) or must be obtained from other sources (the issuing country, the ICAO public key directory, etc.).

The signed data is a special structure containing hashes of all present datagroups in the passport. Integrity of each file can be verified separately (i.e., first the digital signature in EF.SOD is verified and then the integrity of each file is checked by matching its hash against the hash stored in the EF.SOD file).

The digital signature is one of the most important security mechanisms of electronic passports – if not the most important one. Every country chooses the signature scheme that best meets its needs from an implementation and security perspective (supported schemes are RSA PKCS#1 v1.5, RSA PSS, DSA and ECDSA in combination with SHA-1 or any of the SHA-2 hash functions). Every inspection system (InS – a system able to retrieve information from the electronic passport and check/display/use the data) must naturally support all these schemes to be able to verify any valid passport. Signature verification is a relatively simple process, yet complications may arise due to the relatively high number of signature schemes that must be supported, the availability of the root certificates (CSCA) of each country, and the CRLs (each country must issue one at least every 90 days).

It is clear that a digital signature cannot prevent identical copies of the passport content (including the EF.SOD file with digital signature) from being made – so-called cloning. It still makes sense to inspect the classical security features (security printing, watermarks, holograms, etc.) and the correspondence between the printed data and the data stored on the chip also needs to be verified.

3 Active Authentication (AA)

Cloning of passports can be prevented by using a combination of cryptographic techniques and a reasonable level of tamper resistance. To do this a passport-specific asymmetric key pair is stored in the chip. While the public key is freely readable (stored in DG15 with a digitally signed hash), the private key is not readable from the chip and its presence can only be verified using a challenge-response algorithm (based on ISO 9796-2). This protocol is called Active Authentication (AA) and is an optional security feature of electronic passports. AA is optional for EU countries and indeed not all countries implement it (Austria, Czech Republic, and Finland are among the countries that do implement AA).

The point of active authentication is to verify whether the chip in the passport is authentic. The inspection system

generates an 8-byte random challenge and, using the INTERNAL AUTHENTICATE command, asks the chip to authenticate. The chip generates its own random string and cryptographically hashes both parts together. The chip's random string and the hash of both parts (together with a header and a tail) are then signed by the chip's private key. The result is sent back to the inspection system, which verifies the digital signature. If the digital signature is correct, the chip is considered to be authentic. Possible attacks might try to exploit weaknesses in the tamper resistance of the chip or may be based on the analysis of side-channels. If you have a genuine passport at your disposal you might also be able to produce a "copy" that talks back to the genuine passport when necessary. For a more detailed description of such a proxy attack see e.g. [2][4].

There are, however, privacy concerns regarding AA passports. If the challenge sent to the chip is not completely random, but rather specifically structured (for example encoding place and time), inspection systems can store the challenge and the signature as proof that the passport in question was at a given place at a given moment. In reality, the fact that the passport will sign any arbitrary challenge at any place means that the evidence value is very limited. Even so, some countries have decided not to implement active authentication in their passports because of this privacy threat.

Passport holders will soon realize that the passport is in fact a powerful smart card. The use of the chip for the digital signature of documents is apparently insecure as the passport will sign anything without additional authentication, e.g., via PIN (moreover, the challenge-response protocol is definitely not a suitable signature scheme). The use of active authentication for user authentication (e.g., a computer logon) may be much more attractive.

4 Basic Access Control (BAC)

Basic Access Control is a mechanism that prevents passport data from being read before it is authenticated by the inspection system (i.e., it prevents the so-called 'skimming'). The authentication keys are derived from data printed in the machine-readable zone of the data page (see Figure 2). The document number, the holder's birth date, and the passport expiration date are used. All these items are printed on the second line of the machine readable zone and are protected with a check digit (optical character recognition is error prone, hence the choice of data fields with check digits). These three entries are concatenated in an ASCII form (including their respective check digits) and are hashed using the SHA-1 function. The hash value is then used to derive two (112-bit 3DES) keys for encryption and MAC authentication. The command GET CHALLENGE is used to obtain the challenge from the chip and then the inspection system and the chip mutually authenticate using the MUTUAL AUTHENTICATE command. The session key is established and further communication is secured using Secure Messaging.

Figure 2: Scanning of the Machine-readable Zone Data.

BAC is based on a standard mutual authentication technique, which is considered to be secure as long as the keys are kept secret. In the case of electronic passports, the keys are not secret in the classical sense as they are derivable from the data printed in the passport, but even so they may prevent random remote reading. This is, however, slightly problematic as the data used to derive the key does not necessarily have much entropy. Although the theoretical maximum is 58 bits and, in the case of alphanumerical document numbers, 74 bits, real values are significantly lower. Let us discuss the particular entries in more detail [3][13]:

- Holder's birth date: one year has 365 or 366 days, theoretical maximum is 100 years, i.e., around 36524 days total (15.16 bits of entropy). The holder's age can be realistically estimated to an accuracy of 10 years (3652 days, 11.83 bits entropy), often even more accurately.
- Day of expiry: maximal validity of passports is 10 years (therefore approximately 3652 days, 11.83 bits entropy). Passports of children may have a shorter validity (typically 5 years). In the immediate future we will be able to make use of the fact that electronic passports have only been issued for a short period of time. To save space we can also use the fact that passports are only issued on working days and the expiration date is directly related to the day of issue.
- Document number: 9 characters are dedicated to the document number. Shorter document numbers must be padded with padding (<) characters and longer document numbers must be truncated. Document numbers consisting of digits only (and the padding character <) allow for a total number of 11^9 combinations (31.13 bits of entropy); if numbers are alphanumerical then the maximum number is 37^9 of combinations (thus 46.88 bits of entropy). These values are only valid when the passport number is truly random. And that is often not the

case. If certain information about the numbering policy of the particular country is known, then the number of combinations and thus the entropy will decrease. Many countries assign sequential numbers to their passports. If we know the date of issue (or expiration date), the number of possible passport numbers is small. For example a country with 10 million inhabitants issues around a million passports a year. If the year of issue and the range of passport numbers are both known, then the entropy drops to 20 bits. If the month of issue and its range of numbers are known, then the entropy drops further to 17 bits. We could go on to single days, but such detailed information will probably not be available to an average attacker. However, not only insiders but also hoteliers and doorkeepers may know a great deal about the numbering policy (and such information will eventually be published on the Internet). It is more complicated in practice, as we must first guess the issuing country and also the type of passport (e.g., service, alien) as different types may have different numbering sequences.

- Every entry is followed by the check digit. The algorithm is publicly known and the check digit does not introduce any new information.

To estimate the (total) entropy, we can take the sum of the entropies of the entries listed above. But that is correct only when the individual entries are independent. We may debate about the dependency of the expiration date of the document on the birth date of the holder as he/she applies for the document when he/she reaches the age of 15 and then almost regularly renews it (e.g., every 10 years). This may hold true for identity cards and is also country-dependent, but this assumption is not valid for passports as they are issued on request at any age. Therefore we omit that relationship. A similar situation holds for the relationship between the birth date and the document number. But dependency between the document number and the expiration date will typically be present. There is only no dependency for completely random document numbers and only then can we use the sum of the entropies. Otherwise some dependency will always be present and it is only a question of how much information the attacker has about the numbering policy. When the attacker has a significant degree of knowledge, the total entropy may decrease remarkably. Also the smaller the number of passports issued in a country, the higher the chance of guessing the document number. For example, in the case of sequential document numbers and a country issuing 1 million passports uniformly over the year, and if the attacker has detailed knowledge of the document numbers issued on particular days, the entropy of the document number can decrease to about 12 bits. Total entropy then decreases from 58 or 74 bits to approximately 32 bits. A brute-force key search can be then mounted against a significantly smaller number of possible keys.

We can distinguish two types of brute-force attack. Either the complete (successful) communication is eavesdropped and we try to decrypt it, or we try to authenticate against the chip and then communicate with it. When eavesdropping on the communication, we can store the encrypted data and then perform an off-line analysis. If the whole communication has been eavesdropped, we can eventually obtain all transmitted data. The disadvantage is the difficulty of eavesdropping on the communication (i.e., the communication must actually be in progress and we must be able to eavesdrop on it).

The derivation of a single key from the authentication data, data decryption and the comparison of the challenge takes around 1 microsecond on a normal PC. A brute-force search of the space holding authentication data with a size of 2^{32} thus takes slightly more than one hour. A practical demonstration of such an attack against Dutch passports was published by Marc Witteman in [12]. His attack utilized additional knowledge about the dependency between the document number and the expiration date and the knowledge of a next check digit within the document number. Similarly in countries where postal workers deliver electronic passports by mail, these workers could remotely read the content of an electronic passport through a closed envelope as they might know the birthday of the recipient and could easily guess the document number and expiry day (because the passport had just been issued).

As we have already said, eavesdropping on the ongoing communication is not such an easy task. The intended communication range of devices compliant with ISO 14443 is 0-10cm. This does not necessarily mean that eavesdropping at longer ranges is not possible, but an attacker would soon have to cope with a low signal-to-noise ratio problem. While the signal from the inspection system (reader) is detectable at longer distances, eavesdropping on the data sent from the chip (transmitted using load modulation) gets harder with every foot of distance. For discussions about the possible ranges for skimming and eavesdropping see e.g. [7][9].

An on-line attack against the chip can search the key space in the same way, but a single verification of the authentication data is significantly slower – we must communicate with the smart card first and then we have to compute the MAC key and MAC code as well. A single verification then takes approximately 20 milliseconds for standard contactless readers and thus the attack is about 10,000x slower than an off-line attack.

We need to realize that BAC does not restrict access to anybody who is able to read the machine readable zone. If you leave your passport at a hotel reception desk, BAC will not protect your data. On the other hand, there is no additional information stored in the chip that is not already printed in the passport (in EU this is even a legal requirement, except for the fingerprints, of course).

There are also other issues related to contactless communication technology where BAC cannot help. First of all it is possible to remotely detect the presence of passive

contactless chips. Secondly, even before the BAC it is possible to communicate with the chip (e.g., to start the BAC). Anti-collision algorithms need unique chip IDs to address the chips. These chip IDs are typically randomly generated each time the chip is powered, but some type A chips use fixed chip IDs which makes it very simple to track them. Similarly, some error codes may leak information about the chip manufacturer and/or model, which might also increase the chances of guessing the issuing state.

5 Extended Access Control (EAC)

EU passports will also store fingerprints (in DG3) as from June 28, 2009 at the latest (indeed Germany has already started issuing passports with fingerprints, on November 1, 2007). Fingerprints are stored as images in the WSQ format (lossy compression optimized for images of fingerprints). As fingerprints are considered to be more sensitive data than facial images (their recognition capabilities are much better), reading of DG3 will be protected by an additional mechanism. This mechanism is called Extended Access Control. Let us now look at possible theoretical principles for protecting sensitive biometric data in passports to have a better understanding of how European EAC was designed.

5.1 Symmetric Cryptography Based Methods

If access control is based on symmetric cryptography [10] then the data in a passport can be either stored unencrypted and access would be protected by symmetric cryptography based authentication, or can be stored encrypted and not protected by any additional access control mechanism.

A symmetric key would have to be different for each passport (to avoid problems when one passport leaks the key). Keys can be either completely random or derived from a master key by a suitable diversification algorithm (e.g., the passport-specific key could be obtained by encryption of the document number with the master key). We need at least one master key per country, plus probably one key for each passport issuer (i.e., region, embassy, etc.) and the key has to be regularly (e.g., monthly, annually) updated (for passports being issued). An inspection system would then need access to all the keys necessary to access all valid passports (i.e., up to 10 years) for a number of countries. In the case of off-line systems, that would mean that a large number of highly sensitive keys would have to be stored in each inspection system (InS) and the compromise of a single InS would affect current and future access to biometric data in all passports valid at the time of the compromise. This situation is easier to manage with on-line systems. The keys would be physically secure; instead we would have to protect access to the central system. In the event of unauthorized access to the central server, recovery is relatively easy – it is enough to stop the unauthorized access.

The advantage of symmetric or encryption based authentication is the low computational power required of the chip. Basic access control (BAC) is based on shared symmetric keys; we could design a similar protocol based on truly secret keys. When the data on the chip is stored in encrypted form, there are no on-chip computational requirements as the stored data is transparent for the chip and there is no need for any additional access control mechanism. The solution will be secure if the secret keys are kept secret (which is not trivial).

The disadvantage of symmetric methods is the high number of keys that have to be kept secret (and in the case of off-line systems they have to be kept secret at each InS as well). Moreover, secret keys have a long validity period and cannot be revoked. Gaining access to such keys would mean having access to all valid passports which have been issued so far (naturally only for those countries that the compromised InS would be able to access). A clear disadvantage of encrypting the data but not protecting access is the possibility of off-line brute-force (possibly even parallel) attacks. This would be a significantly stronger weapon than an on-line guessing. However, this should still remain just a theoretical threat for a solid encryption algorithm combined with a sufficient key length.

5.2 Asymmetric Cryptography Based Methods

Another way to authenticate the InS is by using PKI. The aim is to reduce the number of secret (private) keys on the inspection system side and to limit the possibility of misuse in the event of compromise. Although there could be several alternative ways to implement Extended Access Control with the help of asymmetric cryptography and PKI, we will follow the proposal of the German BSI [1], which went on to become the European EAC protocol.

5.3 Terminal Authentication

Each country establishes a CV (Country Verifying) certification authority that decides which other countries will have access to sensitive biometric data in their passports. A certificate of this authority is stored in passports (issued by that country) and it forms the initial point of trust (root certificate) for access control. Other countries wishing to access sensitive biometric data (whether stored in their own passports or in the passports of other countries), must establish a DV (Document Verifier) certification authority. This authority will obtain certificates from all countries willing to grant access to the data in their own passports. This DV CA will then issue the certificates to end-point entities actually accessing the biometric data – the inspection systems. See Figure 3.

Each passport stores a CVCA certificate of the issuing country (e.g., the Czech Republic). If an inspection system (e.g., a Spanish one) needs to convince the passport that it is authorized to access sensitive biometric data, it must provide the DV certificate (the Spanish one in our case) signed by the issuing CVCA (Czech) and its own InS certificate (for that particular InS) signed by the DV certification authority (i.e., the Spanish authority in this case). After

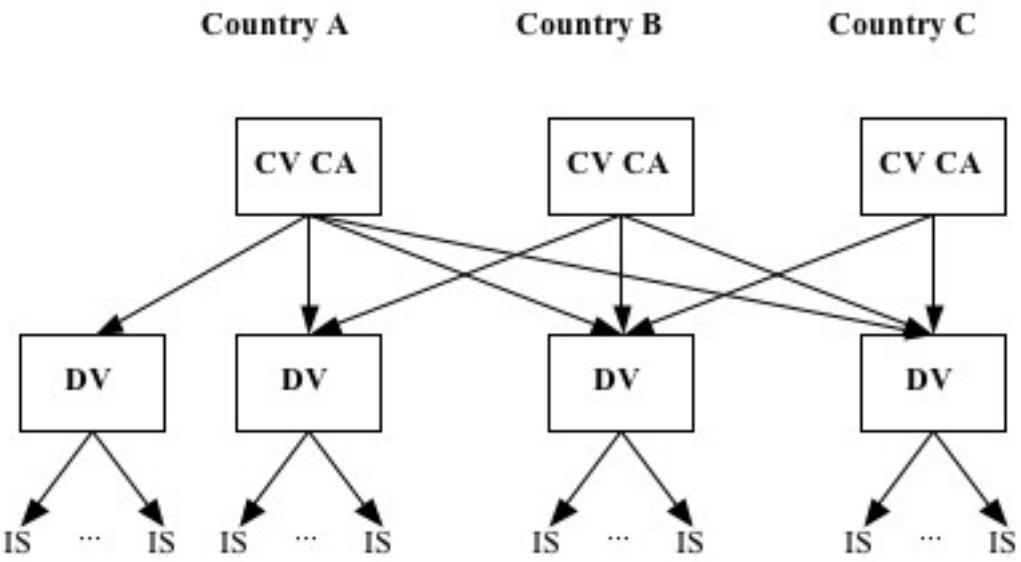


Figure 3: A Simplified View of an EAC PKI Hierarchy.

the passport verifies the entire certification chain it has to check whether the inspection system is allowed to access the corresponding private key. This is performed using a challenge-response protocol. If authentication succeeds, the inspection system can access sensitive biometric data (the DG3 and/or DG4 files). This part of the EAC is called the Terminal Authentication (TA).

The above mentioned process can be slightly more complicated as CVCA certificates are updated from time to time (by link certificates) and bridging link certificates have to be provided (and verified by the passport) first. Terminal authentication can be based on RSA (either PSS or PKCS#1 v1.5 padding is possible) or ECDSA, both in combination with either SHA-1 or one of the SHA-2 variants.

Certificates are sent by using the Manage Security Environment – Set for verification – Digital Signature Template command and the Perform Security Operation – Verify Certificate command. The certificate chain may contain also link certificates if necessary and, after they have been verified, the passport updates the CVCA certificate with a new one (due to a possible overlap of the validity periods of the CVCA certificates, there may be up to two certificates valid at the same time – in such an event both are stored in the passport). Remaining certificates (the DV certificate issued by the CVCA and the DVCA certificate issued for InS) are stored only temporarily and used during the verification of the certificate chain. Once the chain verification succeeds, the passport obtains the public key of the InS and its access rights. Only two access rights are specified at this moment; read access to DG3 (fingerprints) and read access to DG4 (iris image).

After obtaining the public key of an InS we need to verify whether the InS also has access to the corresponding private key. This is done by using a challenge-response protocol. First the inspection system receives an 8-byte long random challenge (using the GET CHALLENGE command), and signs it. In fact what is signed is the concatenation of the passport number, random challenge, and the hash of the ephemeral DH key of the inspection system (from the previous chip authentication). The signature is then sent to the chip for verification using the EXTERNAL AUTHENTICATE command. If verification is successful, the inspection system is authenticated and may access DG3 or DG4 according to the rights assigned. Terminal authentication is not a mandatory part of the communication with an electronic passport. The inspection system can skip terminal authentication if there is no need to read the secondary biometric data from the chip. The InS can be completely offline, e.g., a handheld device storing its own InS key pair and relevant certificates, or there could be a 'central' networked InS providing cryptographic services to a group of terminals at the border (hence Terminal Authentication). Whether InS is offline, online, or a combination of both is up to each individual country.

As the computational power of smart cards is limited, simplified certificates (card verifiable or CV certificates) are used instead of common X.509 certificates. The matter of verification of certificate validity raises an interesting point. As the chip has no internal clock, the only available time-related information is the certificate issue date. If the chip successfully verifies the validity of a given certificate issued on a particular day, then it knows that this date has already passed (or is today) and it can update its own internal time estimate (if the value is newer than the one already stored). It is clear that if a CV CA or DV CA issues (either by mistake, intentionally, or as a result of an attack) a certificate with an issue date in the distant

future, the passport will then reject valid certificates and will become practically unusable. For that reason, only the CVCA (link certificates), DV and domestic InS certificates are used to update the internal date estimate.

5.4 Chip Authentication

In addition to terminal authentication, the European EAC also introduces the Chip Authentication (CA) protocol, which eliminates the low entropy of the BAC key and may also replace active authentication, as access to the private key on the chip is verified (the public key is stored in DG14 and is part of the passive authentication).

An inspection system reads the public part of the Diffie-Hellman (DH) key pair from the passport (the classic DH described in PKCS #3 and DH based on elliptic curves (ECDH) in accordance with ISO 15946 are both supported), together with the domain parameters (stored in DG14). The inspection system then generates its own ephemeral DH key pair (valid only for a single session) using the same domain parameters as the chip key, and sends it to the chip using the Manage Security Environment – Set for Computation – Key Agreement Template command. Both the chip and the InS can then derive the shared secret based on available information. This secret is used to construct two session keys, one for encryption and the other for MAC, that will secure the subsequent communication by Secure Messaging, and SSC (Send Sequence Counter – the message counter value used for protecting against replay attack) is reset to zero. Only after sending and receiving the next command correctly protected with the new session keys can it be known whether chip authentication was successful or not.

As a result of this process a new secure channel is established (low entropy BAC keys are no longer used) and chip authenticity is verified (active authentication is not necessary, but it can be supported by the passport to allow chip authenticity verification by inspection systems that are not EAC-specific and only recognize worldwide ICAO standards).

Worldwide interoperability is not necessary for EAC as sensitive data should be accessible only when there are agreements between countries. Then it is up to the countries to agree on technical details (naturally within the boundaries set out by ICAO standards). The current leader in the EAC field is the EU which designed a protocol for EAC (the protocol was actually designed by the German Federal Office for Information Security).

It is assumed that the protected biometric data will be initially accessible only among EU member states. There has already been some speculation about involving countries such as the USA, Canada, and Australia in the European extended access control system. Looking at the PKI structure of the EAC it is clear that is up to each member state to decide which other countries will have access to data in member states' passports.

While chip authentication replaces active authentication and also improves the security of Secure Messaging, chip and terminal authentication protocols are not standardized by the ICAO at this moment. Hence these protocols will be used only when both the passport and the inspection systems support them. If the passport (e.g., first generation passport) or inspection system (e.g., non-EU or even some older EU systems) do not support the protocol, then we need to fall back on common protocols standardized by the ICAO in Doc 9303 (i.e., BAC and AA). Also, some other countries (outside EU) may not consider fingerprints and iris images to be particularly sensitive data and so data groups DG3 and DG4 in their passports will not be subject to additional protection.

6 Conclusions

It is clear that passive authentication ensuring the authenticity of data stored in electronic passports benefits the security of the electronic part of the passport. But it can only be effective if the Country Signing CA certificates are available at all inspection points. The primary channel for exchanging CSCA certificates is diplomatic post, but it seems that this mechanism is not actually flexible enough. Therefore certificate distribution needs to be improved. There are several proposals; one of them is to use the ICAO public key directory (PKD), initially designed for DS certificates (typically stored in passports anyway) and CRLs also for the distribution of CSCA cross-certificates (one country cross-certifies CSCA certificates of other countries).

While the BAC can prevent basic skimming, the low entropy of the authentication key is its greatest weakness. Efforts to include the optional data field from the machine-readable zone in the key computation (i.e., to increase the entropy) were rejected by ICAO so as not to jeopardize interoperability with existing systems. The only way to improve the strength of BAC is to use random alphanumeric document numbers. Some countries have already changed their numbering policy in order to make attacks against BAC more difficult (e.g. Germany since Nov 2007 [14]). If you are worried that an attacker could communicate with your passport without your knowledge and either try to break the BAC or at least guess some information about the chip, just store your passport in a shielding cover. These covers are now widely available, e.g. [15].

Active authentication preventing passport cloning is implemented by a surprisingly small number of countries. EU passports will prevent cloning by the introduction of the EAC, which includes the chip authentication protocol. EAC also protects access to secondary biometric data and fingerprints (and possibly also iris images) are only readable by authorized border authorities. The key management behind it is not, however, trivial – especially from an organizational point of view. And although DV and IS certificates will have a short validity to limit the use of stolen inspection systems, this will only be effective for passports belonging to frequent travellers.

Disclaimer

The opinions presented are the personal views of the authors and cannot be considered as the official position of the European Commission, where one of the authors is currently working at the Joint Research Centre (JRC) in Ispra, Italy.

References

[1] BSI: Advanced Security Mechanisms for Machine Readable Travel Documents – Extended Access Control (EAC), Version 1.1, TR-03110, <http://www.bsi.bund.de/fachthem/epass/EACTR03110_v110.pdf>.

[2] M. Hlaváč, T. Rosa. A Note on the Relay Attacks on e-passports? The Case of Czech e-passports. <http://eprint.iacr.org/2007/244.pdf>.

[3] J.-H. Hoepman, E. Hubbers, B. Jacobs, M. Oostdijk, R.W. Schreur. Crossing Borders: Security and Privacy Issues of the European e-Passport. <http://www.cs.ru.nl/~jhh/publications/passport.pdf>

[4] ICAO, Document 9303, Edition 6, Part 1, Part 2 and Part 3. <http://www.interoptest-berlin.de/pdf/Kefauver_-_History_of_ICAO_Document_9303.pdf>.

[5] A. Juels, D. Molnar, D. Wagner. Security and Privacy Issues in E-passports. <http://www.cs.berkeley.edu/~dmolnar/papers/RFID-passports.pdf >.

[6] ISO/IEC 14443: Identification cards – Contactless integrated circuit(s) cards –Proximity cards.

[7] I. Kirschenbaum, A. Wool. How to Build a Low-Cost, Extended-Range RFID Skimmer. <http://www.eng.tau.ac.il/~yash/kw-usenix06/index.html>.

[8] E. Kosta, M. Meints, M. Hansen, and M. Gasson. In 1FJP international Federation for Information Processing, Volume 232, New Approaches for Security, Privacy and Trust in Complex Environments, eds. Venter, H-, Eloff, M-, Labuschagne. L., Eloff, J., von Solms, R., (Boston: Springer), pp. 467–472. 2007.

[9] D. Kügler, I. Naumann. Sicherheitsmechanismen für kontaktlose Chips im deutschen Reisepass. Ein Überblick über Sicherheitsmerkmale, Risiken und Gegenmaßnahmen. Datenschutz und Datensicherheit, March 2007.<http://www.bsi.de/fachthem/epass/dud_03_2007_kuegler_naumann.pdf>.

[10] V. Matyáš, Z. Říha, P. Švenda. Bezpečnost elektronických pasů, část II, Crypto-World 1/2007. <http://crypto-world.info/>.

[11] MiniMe (pseudonym), Mahajivana (pseudonym): RFID-Zapper. <http://events.ccc.de/congress/2005/wiki/RFID-Zapper(EN)>.

[12] M. Witteman. Attacks on Digital Passports, WhatTheHack. <http://wiki.whatthehack.org/images/2/28/WTH-slides-Attacks-on-Digital-Passports-Marc-Witteman.pdf>.

[13] Z. Říha. Bezpečnost elektronických pasů, část I. Crypto-World 10/2006. <http://www.crypto-world.info>.

[14] <http://de.wikipedia.org/wiki/Reisepass>.

[15] <http://www.rfid-shield.com/>.