The Practice of Free and Open Source Software Processes

Michel Pawlak, Ciaran Bryce, Stéphane Laurière

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

The Practice of Free and Open Source Software Processes

Michel Pawlak — Ciarán Bryce — Stéphane Laurière

N° 6519

April 2008

Thème COM

ISSN 0249-6399 ISRN INRIA/RR--6519--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
RENNES - BRETAGNE ATLANTIQUE

The Practice of Free and Open Source Software Processes

Michel Pawlak*, Ciarán Bryce†, Stéphane Laurière‡

Thème COM — Systèmes communicants
Équipes-Projets ACES

Rapport de recherche n° 6519 — April 2008 — 39 pages

Abstract: Free and Open Source Software (F/OSS) is widespread today. F/OSS is primarily a movement that permits access to software source code, and, encompasses a software development paradigm that harnesses the efforts of a distributed community. F/OSS is not simply a computer science phenomenon, but also a social, economic and legal one.

The goal of this survey paper is to examine the operational challenges facing F/OSS today, and to review tools, methods and projects that seek to address these. We conclude that the major open challenge is the design of information systems that address community management issues with the same rigor as current tools address content management issues, and that provides support for both F/OSS production and support processes.

Key-words: Free software, open-source software, software licenses.

* Université de Genève, Michel.Pawlak@cui.unige.ch
† INRIA, Ciaran.Bryce@inria.fr
‡ Mandriva, slauriere@mandriva.com

Centre de recherche INRIA Rennes – Bretagne Atlantique
IRISA, Campus universitaire de Beaulieu, 35042 Rennes Cedex
Téléphone : +33 2 99 84 71 00 — Télécopie : +33 2 99 84 71 71

La gestion de processus liés aux projets de logiciel libre

Résumé : Les logiciels libres (F/OSS) constituent une part importante des logiciels utilisés de nos jours. Ils sont produits par une communauté auto-organisée n'ayant aucun point de contrôle centralisé. Une des caractéristiques principales des F/OSS est la mise à disposition des sources du contenu produit (du code, de la documentation, etc.) Tous les utilisateurs de contenu F/OSS re coivent un accès aux sources, l'autorisation de les modifier et d'en redistribuer le contenu selon les conditions définies par la licence employée. Tous les projets F/OSS doivent gérer différents aspects tels que la production du contenu, la gestion des tests, la correction des bugs, la distribution, la documentation, mais également la gestion de la communauté, etc. Nous appelons le *processus* F/OSS le processus englobant tous les mécanismes liés à la gestion de ces aspects. Ce document passe en revue quelques outils et méthodes employés pour gérer le processus F/OSS.

Mots-clés : Logiciel libre, processus de développement, Linux, communauté.

Contents

1 Introduction

Free and Open Source Software (F/OSS) is one of the great facts of software development of the past few years. In this approach, communities of people with common interests collaborate to develop new ideas, models and, *in fine*, produce freely available software. The software is distributed with its source code that can be freely modified by any developer; the modifications made by a developer are, in turn, made available to the community.

In a F/OSS project, the interests of the community push design requirements and specific licenses regulate intellectual property rights. A F/OSS community is responsible for all aspects of software development, from requirements to coding, testing, and even documentation compilation and translation. The F/OSS community is self-organizing; compared to the proprietary software model used by some companies, there is no hierarchal organization of the community. For this reason, F/OSS is said to be organized like a *bazaar*, as opposed to the *cathedral* model of proprietary software development [97].

F/OSS projects have given birth to successful developments such as Linux, Apache, PhP and MySQL. Further, major technology firms such as IBM and Sun Microsystems have become major supporters of this movement with projects like Eclipse [25] and OpenOffice.org [88].

The F/OSS model has even been adapted to non-software domains, fostering community effort to achieve a common goal in a highly collaborative manner. Examples include knowledge sharing (e.g., Wikipedia [121], Wiktionary [123]), health (e.g., Munos Drug Research [79], Finding cures [71], Tropical Disease Initiative [118]) and science [104].

The Challenge of Large Projects

Theoretically, the bigger a project's community, the more successful the project since increasing the community's size brings more ideas and code. Nonetheless, this increase brings a management challenge that needs to be addressed for the well being of the project. For instance, as of 30 October 2006, the standard Mandriva Linux distribution contains 10566 packages[1] so packaging a distribution is thus no easy task. The average package size is 2717431 bytes, so testing packages is a challenge. Further, these packages involve more than one hundred different licenses [69], so reasoning about licensing issues for a system installation is also a challenge. As a further example, Gaim [47], one of the most active projects on SourceForge, had an average of 533 daily read transactions in August 2006, and 17 daily write transactions for an average of 101 files. Thus, maintaining an infrastructure for operating a large-sized project on a daily basis is a challenge.

Large sized F/OSS projects have been known to encounter problems in the following areas:

- **Dependency management** is the problem of identifying and locating the set of packages that need to be installed - or removed - when a given

[1] The F/OSS project's source code is usually distributed as a set of files, where each file contains one or more software modules; each file is generally termed a *package*. This division allows users to optimize downloading by only downloading required packages rather than the whole system. Large projects can have many packages: Debian Linux for instance has nearly 20 000 packages in its distribution.

package is installed. Existing tools such as APT and URPMI, used by major projects, scale poorly to systems with several thousands of packages [1].

- **Testing** is limited as many F/OSS software errors are configuration errors that cannot be detected by the distributor given the multitude of configurations that he needs to test. These errors are only detected after the software is deployed on the clients, which really, is not a satisfactory situation.

- **Code Distribution** is the issue of delivering software to a huge number of end-clients. As the number of users grows, the latency of downloading software from a mirror server increases; this situation is worsened by flash crowd situations when many users attempt download at the same time [2], e.g., at the moment of a new release. Further, the effort needed to keep mirror servers up-to-date is greater when releases are frequent.

There has been much research and development to address these issues. For instance, dependency management algorithms are studied in [27] that are scalable and provably complete. Distribution issues are being addressed with the use of BitTorrent [17] and content distribution architectures [2] that employ peer-to-peer solutions that distribute the network load of downloading among a community of machines.

However, we contend that the **organizational and operational challenges** that F/OSS faces as projects become large are just as important. One such challenge is to ensure that any technical solution, e.g., for dependency management or content distribution, is correctly applied and maintained for all content and community members. Further, consider that F/OSS is more than the simple production, testing and deployment of software; it involves activities such as community management, organization of seminars, production of manuals, starting new projects, etc. A community member may decide to start a new activity inside an existing project, and this can be as varied in nature as fund-raising or server maintenance. The success of a project depends on being able to manage these issues, which requires the following:

- Starting an activity entails *locating competent and potentially interested community members*. Apart from news-groups and mailing lists, there is no way of actively locating potential collaborators, especially for a precise task, e.g. to fix or develop a specialized package or organize a seminar on the project's software.

- *Effort allocation and reallocation* must be gauged since there might be several contributors working on the same topic for the same code package or bug fix, unaware of each other's efforts. Effort reallocation needs support since predicting the workload of contributors is not possible – since there is no developer "contract" in F/OSS, there is no guarantee that a job to be done will indeed get done.

- *Information about content, activities and participants' needs to be shared.* For instance, a user seeking to install a package must be immediately informed of a newly detected configuration error. A member starting a project branch specializing in security must be able to learn of the presence of security experts in the community.

The common element of these requirements is **information availability**. In some cases, pertinent information *may* be held by some contributor, but as contributors are free to leave at will, there is no guarantee that this information makes its way into the community. Further, there is no **standardized method** to collect or retrieve information, nor a guideline indicating what information needs to be collected and made available by F/OSS tools. It is the role of a F/OSS **process** to address both of these issues. A process defines the framework for the production and control of content, the management of community activities, and the accessibility of community and content information whenever it is needed. A well-defined process for a F/OSS project addresses the requirements just set out, and is thus essential for a project to employ one in order to scale as the community grows.

Another role of a F/OSS process is to support project artifact correctness. For instance, each package meta data must include a software license and a complete list of patches and dependencies. The process should render it impossible for the community to create packages that do not have the complete set of meta-data. Further, a formalized description of a process is also the starting point for reasoning about, and improving, the efficiency of the process by permitting information flows and workflow relations to be studied and perhaps compared with processes of other projects.

Goal of paper

The goal of this paper is to examine in more depth the challenges faced by F/OSS processes to help their projects to successfully grow, and to examine tools and techniques that help.

Section 2 gives background into F/OSS. It explains its history and the different approaches. Section 3 pinpoints the operational challenges that need to be addressed by F/OSS projects, and examines why these challenges differ to those of proprietary software development. Section 4 surveys tools, methods and projects that seek to address these challenges, and Section 5 concludes.

2 Free and Open Source Software

The *Free and Open Source Software* (F/OSS) model is a philosophy and methodology characterized by development and production practices that support access to the sources (source code, documentation, etc.) of produced content, and, by the authorization provided to users of the sources to modify and distribute them under specific conditions.

Two different movements put forward this philosophy: Open Source Software (OSS) [85] and Free Software (FS) [38, 40]. Though the movements have different origins and motivations, which we describe later in this section, we group them under the umbrella name F/OSS.

F/OSS encapsulates ideas that have revolutionized the software domain. These include user expectance for software, e.g., making software available to anyone and retrieving software upgrades whenever needed. F/OSS has also revolutionized software production practices by enabling users themselves to improve software, e.g., adapt it to their needs, correct bugs, etc. F/OSS supports maintenance, even when the company that produced the software has gone out of

business. These features are no longer user wishes, but are considered as basic rights by the F/OSS movement and set the background for so-called *software freedom*.

The F/OSS philosophy is driven by the desire to make software widely available in order to enable its fast improvement, to foster innovation, and to enable increased adaptability in highly changing software and hardware environments. To achieve this goal, a community of users forms around the project to pool competence and shared interests. However, as nobody likes putting work into a program and then see someone else gain exclusive (financial) benefit from it, the following rules are respected by the community [94]. Everyone has the right:

- To make copies of a program and to distribute these copies.
- To have access to the software's source code.
- To make improvements to the program, and to distribute these.

2.1 History and Philosophy

The history of Free and Open Source Software started with the UNIX operating system. In 1969-1970, Kenneth Thompson, Dennis Ritchie and others at AT&T Bell Labs began developing a small operating system named Unix on a PDP-7. In 1972-1973, the system was rewritten in the C programming language, a visionary step that led to Unix being the first widely used operating system that could switch from, and outlive, its original hardware.

While AT&T continued developing Unix under the names *System III* and later *System V*, the academic community led by Berkeley, developed a UNIX variant called the Berkley Software Distribution (BSD). Over the years, each variant adopted many key features of the other. Commercially, System V won the war of the standards by getting most of its interfaces into the formal standards, and most hardware vendors switched to it. The BSD branch did not die, but instead became widely used by the research community and for single-purpose servers.

In the PC context, Bill Jolitz's operating system, 386BSD, departed from BSD as a return to UNIX origins but in modern form. It was first released inside the University of California and the US Department of Energy in 1989. Major parts were released in the Networking II (Net/2) release that was labeled as freely distributable by the University of California.

In early 1993, the last 3 coordinators of 386BSD's patch kit – Nate Williams, Rod Grimes and Jordan Hubbard – created the FreeBSD Project [43] after the abandon of 386BSD. Around 1994, Novell and U.C. Berkeley settled a long-running lawsuit over the legal status of the Berkeley Net/2 tape. Novell, who had acquired it from AT&T, owned a large part of the tape. Since FreeBSD was using Net/2, it had to free itself of this commercial part and re-invent itself from a completely new, and rather incomplete, set of 4.4BSD sources from which large chunks of code required for constructing a bootable running system were removed for legal reasons. It took the project until November 1994 to make this transition, which represented the final step to freedom for this operating system.

In respect to software freedom, the reader should keep in mind that when computers first reached universities, they were research tools. Software was

freely passed around and programmers were paid for the act of programming, not for the programs themselves. Only later, when computers reached the business world in the 70's-80's, did programmers begin to support themselves by restricting the rights to their software and charging fees for copies.

The emergence of proprietary software and its implicit restrictions on access to software provoked tensions in the programming community. In 1984, Richard Stallman's Free Software Foundation (FSF) [39] began the GNU project [36] with the aim of creating a free version of the Unix operating system[2]. By free, Stallman meant software that could be freely used, read, modified, and redistributed. The quip often used to clarify the term "free" is to equate it with *free speech*, rather than with *free beer*.

While the FSF was promoting complete freedom, it had in the 1990's [37] trouble developing a free and complete operating system kernel, the major component needed to provide a free operating system. In 1991, this issue was solved when Linus Torvalds began developing an operating system kernel that he named *Linux* [117]. When Linux became free in 1992, its combination with GNU components, BSD components and MIT's X-windows software resulted in a complete free operating system.

The Open Source Definition [85] started life as a policy document of the Debian GNU/Linux Distribution, which was built entirely of free software. However, since numerous licenses purported to be free, Debian had some problem defining what was exactly free and had to make its free software policy clear. These problems were addressed by proposing a Debian Social Contract and the Debian Free Software Guidelines in July 1997. The Social Contract documented Debian's intent to compose their system entirely of free software, and the Free Software Guidelines made it possible to classify software into free and non-free easily, by comparing the software license to the guidelines.

At the beginning of 1997, Eric Raymond was concerned that business people were put off by Richard Stallman's freedom approach, which was very popular among more liberal programmers. He felt that this was hampering the development of Linux in the business world while it flourished in research. In 1998, Netscape announced that it planned to release its browser Navigator as an open source project, and Raymond was invited to help them plan this action. Raymond used this opportunity to sell the free software idea strictly on pragmatic, business-case grounds: the ones that motivated Netscape. This created a precious window of time to market the free software concept to business people and to teach the corporate world about the benefits of an open development process. The principal benefit touted was rapid software development, and that this could be achieved through collaboration with individuals and other companies who shared a common interest in the software.

While the Debian Free Software Guidelines were the right document to define Open Source, they needed a more general name and the removal of Debian-specific references. Bruce Perens edited the Guidelines to form the Open Source Definition – a bill of rights for the computer user. It defines certain rights that a software license must grant the user to be certified as Open Source. Then a certification mark, a special form of trademark meant to be applied to other people's products, was registered. Eric Raymond and Bruce Perens have

[2] GNU stands for "GNU is Not Unix".

since formed the Open Source Initiative (OSI) [86], a non-profit organization, exclusively for managing the Open Source campaign and its certification mark.

As of 1^{st} February 2007, there are 140,417 registered projects and 1,498,326 registered users on SourceForge.net [107] the world's largest Open Source software development web site. Table 1 lists some of the most important open source projects available today. Linux remains the flagship open source development; the major Linux distributions today are Red Hat, Mandriva, Debian and Suse; the major BSDs are FreeBSD and NetBSD.

Project Name	Description of produced software
Ajax	Interactive Web Applications
Apache	HTTP Server
Azureus	BitTorrent client
Eclipse	Extensible Integrated Development Environment
Firefox	Web browser
Gaim	Messenging
GCC	C Compiler
Hibernate	Persistence and query service
Jboss	Application server
KOffice	Office Suite
MySQL	Relational Database
Openoffice	Office Suite
PHP	Hypertext Preprocessor
The Gimp	Photo retouching, image composition and image authoring
Thunderbird	Mailer
Tomcat	Servlet container

Table 1: Examples of F/OSS Projects.

Different approaches; a common philosophy

FSF Richard Stallman has popularized Free Software as a political idea since 1984, when he formed the Free Software Foundation (FSF) and its GNU Project. The philosophy of FSF is that people should have, and appreciate, freedom. To gain freedom, a set of rights was defined that Stallman felt all users should possess. These rights are codified in the GNU General Public License (GPL). It gives users four main "freedom" rights: the freedom i) to run a program for any purpose, ii) to study how a program works and adapt it to their needs, iii) to redistribute copies to help the community and iv) the freedom to improve the program and release this new version to the public. Stallman developed initial works of free software such as the GNU C Compiler and GNU Emacs. His work inspired many others to contribute free software under the GPL.

FreeBSD The goals of the FreeBSD Project are to provide software that may be used for any purpose and without strings attached. For FreeBSD, the first and foremost mission of F/OSS is to provide code to anyone for any purpose, so that the code gets the widest possible use and provides the widest possible benefit. Thus the FreeBSD License permits the redistribution and use in source and binary forms with and without modification, as long as the source code

retains the license itself and a liability disclaimer, and as long as redistributions reproduce these elements in the documentation and/or other materials provided with the distribution. The FreeBSD license is less restrictive than the GNU General Public License (GPL) or Library General Public License (LGPL)[3] since it allows developers to sell code they obtain under the license.

OSI The purpose of the Open Source Initiative (OSI) [86] is to lobby the commercial world with the idea that open source is a rapid and evolutionary process that produces better software than the traditional closed model. OSI also argues that open software is better since, in the closed proprietary world, only a few programmers can see the sources and client companies must blindly trust these sources. Although it is not promoted with the same evangelical fervor as free software, the Open Source Definition (OSD) includes many of Stallman's ideas, and can be considered a derivative of his work. The OSD is a list of ten criteria that software licenses must comply with to be considered as Open Source licenses [85]: i) free redistribution must be ensured; ii) unobfuscated source code must be provided; iii) derived works must be allowed; iv) integrity of the authors source code must be ensured. Further, in order to get the maximum benefit of the process, v) no discrimination against people or groups can be tolerated; vi) no discrimination against fields of endeavor is permitted, this in order to prohibit traps that prevent open source being used commercially; vii) the distribution of license defines that the rights attached to the program must apply to whom the program is distributed; viii) license must not be specific to a product; ix) it cannot restrict other software, and x) it must be technology neutral. Multiple licenses comply with this definition [84], including the GPL.

2.2 The Garage, to Enterprises, to Public Administrations

The F/OSS world has experienced many changes over the years. Long seen as a development model opposed to traditional enterprise development models, it was considered in the enterprise environment as a threat to proprietary software. However, since the release of the Netscape source code under an Open Source license, the border separating F/OSS from enterprise environments has blurred. For instance, shared source licenses such as Microsoft's Shared Source Initiative [76] or Sun Microsystems's Community Source License [111] are a step towards giving access to sources to end users and providing them with benefits not available under traditional proprietary software licenses. This license family defines restrictions ranging from the most restrictive (e.g., content can only be viewed) to the least restrictive (e.g., content can be viewed, modified, or redistributed as source code for either commercial or non-commercial purposes) [122].

A number of large F/OSS projects were initially developed by, or have as partners, well-known companies. For example, Mozilla [78] was a fork of Netscape and its current partners are IBM, Sun Microsystems, Hewlett Packard and Red Hat. The Eclipse platform [25] was originally released into Open Source by IBM and is now a Foundation supported by 50 member companies; it hosts

[3]LGPL is the *Lesser GNU Public License* is a less restrictive variant of the GPL. One of the restrictions of the GPL is that other code that gets linked to GPL code must also be distributed under the terms of the GPL; LGPL removes this particular restriction.

4 major F/OSS projects and 19 sub projects. Project JXTA [60] started as a research project at Sun Microsystems to support peer-to-peer collaborative environments. Today, JXTA covers 117 projects in 6 categories and about 27 enterprises and 29 universities have contributed. The office suite OpenOffice.org [88] has been released into F/OSS by Sun Microsystems, which remains the primary contributor.

Eclipse is a good example of successful synergy between the enterprise and F/OSS worlds. IBM takes advantage of inputs from the community to improve the platform, and sells its Websphere product built on top of Eclipse. In the meantime, the community provides new plug-ins. Other companies also provide plug-ins and tools that integrate with the platform and IBM products. This situation allows all parties to benefit from the base platform.

F/OSS and commercial cooperation needs legal regulation to work smoothly. Thus, the Eclipse project is released under the terms and conditions of the Common Public License (CPL) [19]. This license specifies that *each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free patent license* in order to protect contributors from being sued for patent infringement when using code under the CPL. The CPL also specifies that only commercial contributors, i.e. contributors reselling products released under the CPL, are liable for damages caused by the product.

Various F/OSS projects now exist to support enterprise environments. For instance, Open For Business (OFBiz) seeks to provide applications and tools to easily and efficiently develop and maintain enterprise applications [7]. The growing availability of open source products such as enterprise resource planning (ERPs) and consumer resource management (CRMs) [108, 81, 92, 18] is another sign of the enterprise - F/OSS integration.

European Public Administrations increasingly rely on F/OSS. In France for instance, public sector institutions increasingly use F/OSS solutions for their IT systems since 1998. Concerned sectors include the Ministry of Defense, the Ministry of Justice (e.g., the national crime register), the Ministry of Economy, Finance and Industry. Malaysia has an Open Source Awareness Programme for educating and assisting public sector users in adopting and implementing F/OSS solutions [67]. The Open Source Observatory of the IDABC [53] (Interoperable Delivery of European e-Government Services to public Administrations, Businesses and Citizens) offers a good overview about OSS-related government activities in Europe and abroad.

3 The Challenges for F/OSS

This section looks at the specific features of F/OSS, and at how it differs from proprietary software content and production practices. Understanding these features is important to run F/OSS processes efficiently, and in particular, to ensure that the processes can work gracefully as the project increases in size. The first subsection examines the distinguishing features of F/OSS environments; the second subsection provides process guidelines for content, community and process management.

3.1 The Distinguishing Features of F/OSS

There are several features of F/OSS software and communities that distinguish them from their proprietary counterparts.

1 - The License The distinguishing feature of content produced by a F/OSS project is the license. It defines the obligations and rights concerning the use, modification, and further distribution of content.

A large amount of licenses can be used for Open Source. The Open Source Initiative (OSI) [86] provides a set of approved licenses, the purpose of which is to provide the community with a reliable way of knowing if a piece of software offers the qualities expected of open source software. As of September 2006, 56 licenses are approved by the OSI [84]. Similarly, the Free Software Foundation defines a list of licenses and classifies them as GPL-compatible, GPL-incompatible or Non-Free Software Licenses [41]. The different types of F/OSS licenses are discussed in [61].

The multiplicity of license raises the issue of license compatibility in software projects. This is particularly important for projects like the Linux distributions that use packages from independent projects that can have different licenses. Not every license can be combined with each other; this is the case with the General Public License (GPL) [42] which is incompatible with many other licenses. While license compatibility issues can occur when projects use content produced in independent projects, it can also occur within single projects using different licenses or when integrating Open Source Software with an existing information system.

2 - Absence of Standards Information in F/OSS projects describing content and community is not standardized. For instance, there is no globally adhered to rule indicating what metadata should be bound to packages, what precisely a patch is, etc. As a result, it is difficult to build generalized F/OSS tools. From the process management perspective, every project can have its own definition of the responsibilities assigned to a *tester* or *distribution manager*, and thus, even activities like *testing* and *distribution* can require different responsibilities and competence in different projects.

3 - High Decentralization F/OSS resources are highly distributed. For instance, contributors to the Mandriva Linux distribution come from several countries. The Debian Linux distribution maintains a page indicating the different locations of their developers who are willing to provide this information [21]. Figure 1 was generated using the program xplanet [125] from the list of coordinates found on this page. It is obviously difficult for F/OSS contributors to meet each other. In fact, they rarely meet or know each other, and communicate using the tools like e-mail, mailing lists, Wikis and other tools provided by projects.

4 - Dynamic Project Structures New F/OSS projects are regularly created, and existing projects can take new directions. A project may decide to develop new applications. The community involved in a particular F/OSS project evolves as well: developers join and others get promoted to committers. These roles are not full-time jobs; for instance, committers can take time out to test packages, to cater for a new release, or may even leave the project in order to start a new one. F/OSS projects do not have a precise number of participants; they often start small with a few motivated people and may become huge involving hundreds of participants.

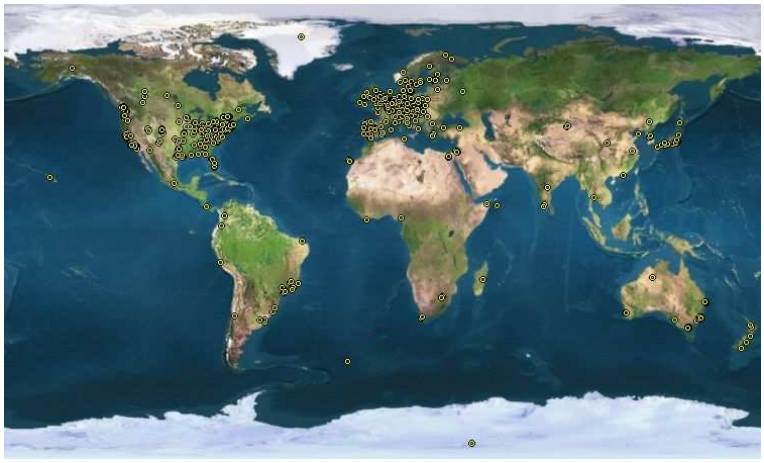

Figure 1: Map of Debian developers locations

In proprietary software, upgrades are provided through major versions, and critical bug fixes are provided on a punctual basis. In contrast, the whole F/OSS philosophy is based on dynamics. Indeed, the evolution of F/OSS is driven by the multiple updates, problem reports and bug fixes contributed by the community. While the closed source software approach seeks to hide these intermediate milestones from the public, F/OSS shows them in the hope that some contributor will handle them as fast as possible. For instance, the Debian project's social contract stipulates that the project "will not hide problems" [23]. The Eclipse project [25] provides in addition to its *stable build*, an *integration build* integrating most recent changes to the project on a regular basis, and *nightly builds* providing the changes made day by day. One of the most active projects on SourceForge, Gaim [47], had an average of 8 bugs opened by day and 9 bugs closed by day between November 2005 and November 2006.

5 - The Challenge of Quality The set of quality problems related to F/OSS are now well identified [74]. These problems include unsupported or orphaned code, configuration management issues, security updates management, lack of documentation, difficulty to attract volunteers, and communication and coordination issues. Most of these quality problems are bound to a lack of efficient process management within the project. This shows that while developers may understand that process management is important for quality assessment, they do not necessarily have the experience in it. The study [74] also showed differences between the development practices and processes in surveyed projects, and suggests that few projects adopt all required processes for quality.

Quality is not facilitated by the impressive growth rate of some large projects. For instance, a study of the evolution of the stable versions of Debian from 1998 onwards [99] shows that the distribution doubles in size (measured by number of packages or by lines of code) approximately every two years. This result, combined with the huge size of the system (about 200 MSLOC and 8000 packages in 2005), may lead to significant management issues in the future. Indeed, since the number of packages is growing linearly, and since the specifics of each package imposes a limit on the number of packages per developer, this

means that projects also need to grow in terms of developers at the same pace. However, such growth is not easy and causes problems of its own, specifically in the area of coordination. This has probably influenced the delays in the release process of recent stable versions of Debian.

6 - Contributor Identity and Interest F/OSS projects exist because people contribute code, documentation and services (e.g., testing, translation of manuals, etc.). Notably, contributors are not directly paid by the project for their contribution. Some contributors might be indirectly paid by a foundation supporting the project or by their company, which, for strategic reasons, supports the F/OSS process. Generally though, contributors are people with no financial interest in the project, but who contribute because they believe in the F/OSS philosophy and projects to which they contribute. People also contribute since they enjoy being a member of a community; this explains why F/OSS is also considered to be a social phenomenon.

F/OSS environments are often misunderstood as being completely anonymous since anybody can contribute to projects without identifying himself. While anonymity is desired in some cases, full anonymity is not always authorized. Anonymous access is often allowed to many services like `ftp`, or content management systems like subversion for content reading. However, in cases where code sources or binaries are subject to license restrictions, identification might be required, even for read access. In order to obtain write access, users might have to be identified for security and information integrity reasons. In FreeBSD for instance, mutual identification of committers with the CVS code repository servers [20] is required through the use of public key cryptography.

An analysis of the Debian project provides insight into how volunteer involvement affects released software and the developer community itself [98]. The study has three conclusions: i) globally, the involvement duration of volunteers is stable; ii) experienced volunteers seek increased responsibilities and increase their involvement over time; and iii) the voluntary effort is stable even when volunteers leave, since tasks are taken over by other volunteers. However, the report also highlights the worrisome trend of a growing number of packages per maintainer. This can create scalability problems as the number of packages in the distribution increases and the project does not grow by a proportional number of developers.

3.2 Requirements for F/OSS Processes

Consider the following citation of Ian Murdock, founder of the Debian project.

> "Linux is not a product. Rather, Linux is a collection of software components, individually crafted by thousands of independent hands around the world, with each component changing and evolving on its own independent timetable. ... Linux is not a product. It is a process."

While a F/OSS project is characterized by community and content, the process defines "how things happen". It expresses the actions of community members to produce content and manage the community. The challenge today is to develop information system support for processes, and in particular, to accompany these processes as the projects they manage get larger. The design of

these information systems requires that guidelines exist for content, community and process; these are looked at in this subsection.

3.2.1 Content Guidelines

Uniformity of Content By content, we mean anything produced by a F/OSS project. This can be source code, binary code or documentation. There is no real difference from the process viewpoint between documentation and code. For instance, some of the OpenOffice developers produce natural language translations of user manuals rather than code; their content gets packaged, tested and linked to others just as code from developers gets treated. The implication is that it must be as easy to create and support (via tools) F/OSS activities around documentation as it is around code.

Integrity of Content To be used correctly by a community, all forms of content must be correctly packaged before distribution. This entails including in the package a precise set of metadata, e.g., date, dependency information[4], license, etc. The purpose of the metadata is to ensure that the package can be used (i.e., installed if it is a code package), modified and republished in the community. A process should not permit the creation of a package if the complete set of metadata is not provided.

Review of Content Community members should be able to search through content using a range of properties. Currently, the need to know package names is an unacceptable burden for both end users and developers. Content retrieval should rather be built around a mechanism that enables actors so specify what they need, e.g., content retrieval based on the license or on the configuration of the client's machine – an operation currently not possible.

Further, for any unit of content, it must be possible to determine its origin. This information includes the original development source and also the evolution that the content underwent (versions, branching, etc.). For security, this information must be indelibly bound to the content.

3.2.2 Community Guidelines

Projects live from the contributions made by members of their communities. Managing a community implies being able to harness the competence that exists in the community – this is the fundamental difference between successful and less successful projects. Harnessing the potential of the community means that the community support process is able to profile the community in the following ways.

Contributor overview Project managers should have access to a complete overview of their project's community members. Information available should include the number of participants in the project, the interests, knowledge and competence of each actor, the responsibilities that each of them is assigned, the achievements made by each member in the scope of the project, and thus

[4]This captures several types of relation between packages. Build (or source) dependencies define the prerequisites for building and compiling a package. Binary dependencies specify the prerequisites for installing a package. Configuration dependencies provide the prerequisites for configuring a package once it is installed. Conflicts specify the incompatibilities between packages at installation and configuration levels. Finally, replacements define the relations between packages that may replace each other's functionality, e.g., the mail server sendmail [106] can replace the mail server exim [29].

the quality of work done by each. This overview is useful to be able to **locate contributors** based on interests of the community members, their competence or experience. This is particularly the case for specialized tasks, e.g., starting a sub-project that deals with encryption of in-memory data, or locating someone to give tutorials on the project's software in a town.

Workload overview. To avoid situations where community members are overloaded, another aspect of community management is to visualize contributor workload. For instance, an increase in software defects or manual tests to run may give too much work to the set of contributors responsible for testing. Being able to detect such situations can help to react fast by suggesting a reattribution of tasks to contributors.

Role Management The purpose of a role is to define the operations associated with a given task, and also the security privileges and responsibilities associated. In the context of a patching activity for instance, roles can be defined to correspond to new patch submission and to patch committing. Working with the role abstraction permits project managers to reason about the work done, or to be done, in a project without being concerned about who is currently attributed to these roles. It is for this reason that roles have been so widely used in the security community [102]. Further, the mapping from contributors to roles can be analyzed to ensure that contributors only have roles needed for executing the tasks they have been assigned, and avoids the risk of forgotten rights where users have privileges that should be revoked. This is the case since a user's rights are strictly defined by the roles he is attributed – removing a role removes associated rights.

3.2.3 Process Guidelines

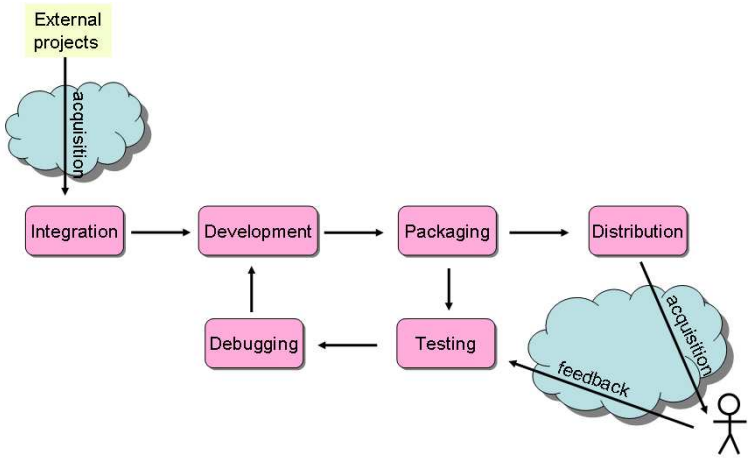

Figure 2: Example of production and distribution process

As suggested in Figure 2, the F/OSS model can be viewed as a series of production processes – documentation, development, testing, integration, packaging, etc. These F/OSS sub-processes are highly inter-dependent. For instance, consider improving the distribution process with an elaborate search tool that

allows users to specify search requests for content using natural language constructs. Integrating the tool is not simply an issue for the distribution process; rather, it requires expressiveness metadata in the content package, and content packagers can only add this. There are also workflow dependencies between processes, e.g., content cannot be tested before it has been produced, it cannot be packaged before a testing process certifies a given stability level, and it cannot be distributed as long as it has not been properly packaged, etc.

The guidelines just presented in relation to content and community can only be satisfied by a F/OSS process. They entail complementing the production processes with support processes, e.g., for efficient community management (e.g., cataloging the experience and interests of actors), measurement and metrics management (i.e., to analyze the health of a project), and process management (e.g., to identify what has to be done within a project, the recurring tasks, tasks to be affected to community members, etc.). These support processes are often neglected in F/OSS projects, be it for lack of interest in them, the lack of gratification that working on them provides, or due to a lack of knowledge about their importance to the overall F/OSS project outcome.

It is extremely difficult to have an exhaustive list of processes involved in different F/OSS projects, though some research work has been done on automatic process extraction [59]. Extracting and formalizing the process would useful, as it would permit the following list of functionalities.

Streamlining the F/OSS Process is the effort of making project management more efficient. A formal description of a process is clearly useful for this since project management would be able to chain processes, build workflows, measure them, and create rendezvous or synchronization points. The description is also useful for pinpointing superfluous information flows and process steps. The desire to optimize business processes in this way for instance has led to process modeling and execution languages such as BPML [11] and XLANG [116].

Know-how sharing While F/OSS focuses on code sharing, a major stake for F/OSS is to go towards a situation where large projects can share their know-how about process handling, measurement and evaluation with small projects. Process formalization can help here since it is itself a knowledge artifact for sharing: it has a concrete representation that can be exchanged and discussed by project participants.

Process Dashboards provide operational information indicating how the processes are being executed, showing if this execution is in-line with predictions previously made, if unexpected events are happening, etc. Dashboards could also support strategic decisions, by enabling projections and estimations of the impact of decisions such as involving more contributors as testers or distributors. Dashboards are a key element of all processes, and in F/OSS they can encode all information related to content and community (e.g., trace package evolution, test results, contributor patch record, etc.). Thus, if a project depends on an external activity such as a testing service, and if the latter lacks testers to achieve its task, the dashboard should detect this and permit contributors to be moved to the testing activity.

Building interoperable F/OSS projects is important since projects can rely on content produced in independent projects, e.g., Linux distributions. This raises new challenges with regard to interoperability like information format standards as well as standards for descriptions of processes employed by each

project. Interoperability is a process issue since a project needs clear guidelines about working with external projects: i.e., a description of the steps to be taken, the information and formats to be exchanged, the roles involved in project cooperation, etc.

4 Existing Approaches and Solutions

The goal of this section is to review methods and projects that consider the process view of F/OSS projects. The focus of existing work has generally touched two aspects of process management – community organization to a certain degree that we look at in Subsection 4.1, and especially quality assessment, which is the subject of Subsection 4.2. Finally, Subsection 4.3 looks at work on overall process management.

4.1 F/OSS Community Organization

A key characteristic of F/OSS is the self-organization of the community. As most F/OSS projects rely on the work of volunteers, attracting people who contribute their time and technical skills is of paramount importance, both in technical and economic terms. Ensuring that contributors do not neglect their work is another challenge. The Debian approach to this issue [72] employs several approaches such as developer hints, email-based activity confirmation and a quality assurance team. The project recommends preventive measures such as explaining the problem to prospective volunteers, introducing maintainer redundancy to limit the damage of inactive maintainers, and limiting the number of low-interest packages.

To become a registered Debian developer, users have to pass through a process of identity, intentions and technical skills verification [22]. This process is partially run online through the evaluation of patches submitted, but to be completed, it involves that a Debian maintainer signs the GnuPG key of the new developer, which implies that people have to meet in person. Other projects can have other ways of dealing with newcomers. For instance, FreeBSD has a disciplined group of core developers that approves source code committers; similarly, the ports management group approves ports committers, and the same process is applied for documentation. Once approved, new committers are assigned a mentor who works on the same code tree as them and who is responsible for everything their pupils do in the FreeBSD project. The role of the mentor is to answer questions and to review patches submitted by the newcomers. There is no real penalty if anything goes wrong, but it reflects badly on both the mentor and the new committer – this is sufficient incentive to work well together. As time passes, the mentor verification becomes less strict, but he is still responsible for the committer assigned to him.

A good example of what can currently be called advanced community management is Mandriva's *Mandrivaclub* [68]. This club is built around a Wiki where community members can find all documentation they need, participate in forums, gain access to a shared knowledge base, and chat with the Mandriva team. However, as users' motivation, interests or knowledge are not considered by this solution, advanced community management that helps users contribute or helps the project find skilled contributors is not possible. There is still

progress to be made to achieve the community management guidelines outlined in the previous section.

An interesting approach to the issue of community management in F/OSS is to create a F/OSS project whose goal is to facilitate community organization! For instance, the SELF [105] project aims to provide a platform for educational and training material. The first goal of the project is to provide information, educational and training materials on Free Software and Open Standards presented in different languages and formats. The second goal is to offer a platform for the evaluation, adaptation, creation and translation of this material. The production process of the material is based on the organizational model of Wikipedia [121]. The SELF Platform specifically embraces universities, schools and training centers, F/OSS communities, software companies, publishers and government bodies.

The TEAM project (The Tightening knowledge sharing in distributed software communities by applying semantic technologies) [113] is a European Union funded project addressing the need for a knowledge sharing environment specializing in the distributed engineering and management of software systems. The features of the environment include a semantic search tool for relevant knowledge items, a metadata repository for acquired knowledge that incorporates a tool to reason about its consistency, and a P2P Infrastructure for knowledge exchange between Knowledge Desktops and software developers. An interesting feature of TEAM is that it underlines the fact that the value of the f/oss community is not just the *content* produced, but the *knowledge* required to produce it.

The SELF and TEAM research projects are funded by the European Union, which shows that the need for F/OSS community management support is widely recognized at government level. In practice however, little effort is put into community management by the majority of F/OSS projects. Some projects keep in touch with their community members through mailing lists, other choose more advanced means to tackle the community management issue through tools like Wikis, yet most small projects put no effort at all into this task.

4.2 F/OSS Quality Assessment

The F/OSS community often considers the F/OSS model to guarantee a high level of quality by itself, with features such as source code availability supposedly facilitating the detection of errors. In practice, studies confirm that quality can only come from an organized community effort to ensure quality [58].

As the now abandoned ISO 8402 [54] norm defined, quality assurance deals with *planned and systematic activities* for fulfilling requirements for software quality. Project processes impact on the quality of a project and its software. For instance, poor coordination between developers in the development process can lead to latency in code production; this directly impacts on the frequency of distribution releases and also on volunteers' willingness to contribute if they are annoyed by delays. To study quality improvement, orchestrated processes involving tools and specialized metrics are needed both to measure quality and to evaluate quality improvement techniques. A starting point for quality improvement is to examine the project's defect reports; this approach was taken in [75] for Debian with an analysis of over 7000 defect reports.

We break our presentation of QA into two parts. The first looks at the biggest field of research and development – tools for QA of content produced by a F/OSS community. We then look at tools for measuring the quality of a F/OSS process itself.

4.2.1 QA of Content

Quality Assessment tools and frameworks organize tests, create test suites, execute them and keep track of obtained results. Test suites are defined for a specific domain and aim to achieve testing coverage for this domain.

As this subsection illustrates, the set of tools is quite large even if they do not all include each of the functionalities we mentioned for QA. This is generally due to each project developing its own tools, though some, like Bugzilla, are used across projects. Despite the many tools, the general and emerging features of tools are the same.

- **Automate testing** as much as possible, and thus have to rely on fewer volunteers than for development. This is important since testing is a far less attractive activity than development.
- Offer **visualization tools** to the community about the status of content and test reports. This data must be understandable to all types of users, e.g., developers, business analysts and end-users. The data also serves to encourage volunteers for those packages that most need work.
- The ultimate aim, from a process perspective, is that bugs and **tests be handled with the same rigor as code packages**, with the emergence and formalization of process activities to encourage and manage volunteers. Poor management of content QA activities leads, ultimately, to poor quality content.

As the range of available open source testing tools is extremely wide, Opensourcetesting.org [90] provides users with a centralized point of departure. Each tool is described by a profile and categorized accordingly to the family it belongs to. The website splits the tools in two main groups: testing tools and unit testing tools. Each is further sub-categorized: testing category contains tools for functional testing, performance testing, test management, bug tracking, link checking, security, while the unit testing category contains tools for creating unit tests in different languages such as in Ada, C/C++, HTML, Java, Javascript, .NET, Perl, PHP, Python, Ruby, SQL, Tcl and XML.

The broad choice of tools gives projects a lot of flexibility in dealing with Quality Assessment. For instance Mandriva has its own Quality Assurance Lab's Contributor's Corner [70] which aims at centralizing information about how quality assurance is dealt with at Mandriva, so that the community can participate in their public test campaigns and help them improve the quality of the Mandriva Linux distribution. The lab tests, validates and certifies Mandriva's own and integrated software, as well as hardware compatibility. The tools used are Bugzilla for detect management, its companion for testing, Testzilla, as well as a database containing information on all of Mandriva's RPMs, e.g., data on dependencies, changelogs, etc. Nonetheless, despite the broad choice of tools, there are no standardization guiding projects in the choice of the tools for handling quality.

Linux Test Project (LTP) [66] is an example of a test suite. It is a joint project started by SGI and maintained by IBM, whose goal is to deliver a collection of tools for testing the Linux kernel and components that validate the reliability, robustness, and stability of Linux. LTP also gathers test results within a Linux Test Tools Table. This provides the F/OSS community with a comprehensive list of the tools used for testing the various components of Linux as well as a code coverage analysis tool whose aim is to graphically identify the areas in the kernel impacted by the execution of these tests. This enables developers to use the LTP test suite more effectively and also to identify the test contributions needed to improve the test suite.

Bugzilla [13] Bug-Tracking System is used to track bugs and code changes, communicate with teammates, submit and review patches, and manage quality assurance. Information is provided for each bug such as the bug reporter, the version of software, platform, priority, product, component, OS, severity, initial state, the person the bug has been assigned to, the persons who have to be informed of its creation, an url, a summary and description. Having people assigned to bugs helps implement accountability. Registered user can leave comments and propose patches. Bugzilla allows searching existing bugs by status, product, keywords, but one of its most interesting features is its ability to generate precise and complete tabular or graphical reports. Indeed, the user is able to pick any data information existing in the bug database and use it as an element of the report. Examples of usable data are the summary of the bugs, the products that are concerned, the components of the products, their version, the comments, the URL, keywords, bug status, resolution, severity, priority, concerned hardware, related software, email addresses and number of bugs and bug changes. The users define XYZ axis, put filters to the different data, and then generate a report in tabular or graphical form (bar, line or pie charts). Thus any information can be used as an indicator. Furthermore, Bugzilla offers the capability to build bug dependencies and to illustrate them using graphs or trees. Thus bug blocks can be indicated and graphically represented. Bugzilla has also a "voting" feature allowing users to vote for the importance of a bug. All users can be given a certain number of votes and when voting, a user indicates that the bug is of the highest importance and needs to be fixed rapidly. The number of available votes per user for a given product depends on the administrator of that product. Figure 3 gives an example of a chart generated by Mandriva's bugzilla instance mapping the number of active bugs (on February 21st, 2008) to a severity level. Figure 4 shows an extract of a page for a Mandriva bug.

Fitnesse [30] is a lightweight, open-source framework aiming at enhancing collaboration in software development. It allows Acceptance Tests to be collaboratively defined using a wiki. Acceptance tests differ from unit tests in the sense that they define what the code should do from a functional and end user point of view. Indeed, while unit tests provide testing at a smaller granularity than acceptance tests do, and ensure that the code is correctly built, fitnesse test cases define business logic. Acceptance test are expressed as tables containing inputs and expected outputs and these tables are defined by the tester. They are readable by customers and aim at ensuring that the application is doing what it is meant to. Unit and acceptance tests are complementary and both should be highlighted. Tests are created through the wiki and can be directly run from the framework. This is done by triggering so called custom "fixtures"

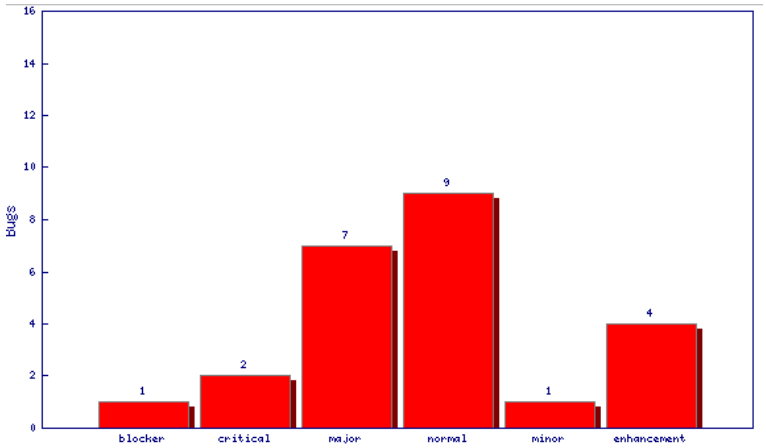

Figure 3: Mandriva Linux bug severity levels

Figure 4: Mandriva Linux bug description

which map customer language constructs to the implementation. Created tests can be organized as test suites and contain cross-references to other test suites.

There are several testing frameworks used by f/oss projects but they do not differ greatly in functionality. The key systems are Testopia [115] (a test case management extension for Bugzilla), RTH [100] (a web-based tool designed to manage requirements, tests, test results, and defects throughout the application life cycle), Test Case Web (TCW) [112] (an online test management system built with PHP and a SQL backend), Test Link [114] (an open source web based test management and test execution system), Salome-TMF [101] (a test management tool aiming to provide an open framework allowing automatic test execution, the production of documents, and the management of defects/requirements using the ISO 9646 [55] definition of tests), the Software Testing Automation Framework (STAF) [109] (an open source, multi-platform, multi-language framework designed around the idea of reusable components), the Open Office Automated GUI Testing Project [89, 110] (a test framework with test scripts and an application (VCL TestTool) to test almost the whole Open Office [88] application automatically), and GNU/Linux Desktop Testing Project (GNU/LDTP) [65] (aimed at producing high quality test automation framework and cutting-edge tools that can be used to test GNU/Linux Desktop and improve it).

4.2.2 F/OSS Project Measurement

The previous section looked at testing tools for the basic commodity of F/OSS projects – content. Although less enshrined in the culture of F/OSS projects, measurement of the project itself is also an area of study. On the one hand, there are models that deal with project measures. On the other, there are projects that collect measures from open source projects, and as such facilitate comparison and the definition of new tools for project measurement.

Project Measures Business Readiness Rating (BRR) [12, 87] is being proposed as a new standard model for rating open source software. It is intended to enable the entire community (enterprise adopters and developers) to rate software in an open and standardized way. The calculation employed in the Business Readiness Rating model weighs the factors that have proved to be most important for a successful deployment of open source software in specific settings: functionality, quality, performance, support, community size, security, and others. The Business Readiness Rating model is open and flexible, yet standardized. This allows for a broad implementation of a systematic and transparent assessment of both open source software and proprietary software. Figure 5 shows an extract of rating data for JBoss.

Navica's Open Source Maturity Model (OSMM) [80] is designed to be a lightweight process that can evaluate an open source product's maturity in a short time (up to two weeks). It assesses the maturity level of all key product elements such as software, support, documentation, training, product integration and professional services. As output, OSMM provides a numeric score between 0 and 100 that may be compared against recommended levels for different purposes; for instance, the measure may be interpreted differently according to whether an organization is an early adopter or a pragmatic user of IT.

Capgemini's Open Source Maturity Model (OSMM) [35] describes how an Open Source product should be assessed to ensure that the product meets the

Evaluated Technology And Functional Orientation			BRR
Jboss			**4,29**
Component type: J2EE Application Container			
Usage Setting: Mission Critical			
Rank Category Title	Weight	Unweighted Rating	Weighted Rating
1 Functionality	22,00%	5	1,1
Rank Category Title	Weight	Unweighted Rating	Weighted Rating
11 Usability	0,00%	0	0
Rank Category Title	Weight	Unweighted Rating	Weighted Rating
2 Quality	18,00%	3,95	0,71
Rank Category Title	Weight	Unweighted Rating	Weighted Rating
5 Security	12,00%	3,5	0,42
Rank Category Title	Weight	Unweighted Rating	Weighted Rating
4 Performance	15,00%	3	0,45
Rank Category Title	Weight	Unweighted Rating	Weighted Rating
3 Scalability	15,00%	5	0,75
Rank Category Title	Weight	Unweighted Rating	Weighted Rating
8 Architecture	0,00%	0	0
Rank Category Title	Weight	Unweighted Rating	Weighted Rating
10 Support	0,00%	0	0
Rank Category Title	Weight	Unweighted Rating	Weighted Rating
9 Documentation	0,00%	0	0
Rank Category Title	Weight	Unweighted Rating	Weighted Rating
7 Adoption	8,00%	4,5	0,36
Rank Category Title	Weight	Unweighted Rating	Weighted Rating
6 Community	10,00%	5	0,5
Rank Category Title	Weight	Unweighted Rating	Weighted Rating
12 Professionalism	0,00%	0	0

Figure 5: Extract of Business Rating Model Data

IT challenges a company faces. The OSMM accomplishes this by linking an extensive product analysis with a thorough review of the company and its IT issues.

Information Availability The relative accessibility of information about open source software makes it an interesting target for quantitative analysis to discover some hidden properties and trends of this software development model. FLOSSMole [56, 33] (Free/Libre Open Source Software Mole and formerly OSSMole) is a data mining project that aims at providing data and reports about existing F/OSS projects and teams. A particularity of FLOSSMole is that it promotes compatibility both across sources of F/OSS data and across research groups. The project gathers, shares and stores comparable data and analysis of F/OSS development for academic research. Having such a collaborative and compatible data and analysis repository enables reproducible, extendable and comparable research on F/OSS. The project provides scripts for analyzing the raw data and provides some tools enabling users to gather their own data. The raw data used by FLOSSMole is donated from other research teams and projects to create common frames of communication. Collected data includes page views, downloads, bandwidth consumed by downloading, and number of comments posted, etc.

The main objective of the FLOSSMetrics (Free/Libre Open Source Software Metrics) Project is to construct, publish and analyze a large scale database with information and metrics about F/OSS development coming from several thousands of software projects, using existing methodologies, and tools already developed [32]. The targets of the projects are to identify and evaluate sources

of data and develop a comprehensive database structure, built upon the results of the CALIBRE Project [15].

Ways in which properties of F/OSS can be acquired have been described in [120]. This research analyzed the largest open source hosting facility – SourceForge – to obtain quantitative information about existing projects. It focused on aspects such as project activity, average number of developers per project, number of language translations per project, etc. Cross-comparisons were done with the results provided by FLOSSMole on the following aspects: number of developers, intended audience, usage of licenses, targeted operating systems, language of implementation, declared development status, registration history and declared topics. Among its conclusions, the analysis showed that the main target audience for F/OSS projects are developers. While fostering developer involvement, this sidesteps the question of growing user communities. Further, the analysis showed that GPL licensing is the most common in the F/OSS world. A last interesting conclusion of this analysis is that a vast majority of open source projects declares their development status at the "unstable" level or even in the planning phase. As only a minority of F/OSS projects ever make it to the level of being popular, and thus successful, this raises questions about the reasons that make a F/OSS project successful or not.

In [51], the authors use publicly available storage (source code snapshots, CVS repositories, etc), as a source for analyzing and characterizing the evolution of F/OSS projects. Since the base information is public, and the tools used are readily available, other groups can easily reproduce and review the results. Obtained characterization is then used as the basis for qualitative analysis including correlations and comparative studies of projects.

Ohloh.net [83] is a resource for open source intelligence on thousands of open source projects. Ohloh collects software metrics from a variety of sources including the project's source code and the software development infrastructure used by the project's development team. It provides information about the developers involved in the project, showing details about their activity (i.e. the frequency on contributions), the languages used by the project, and the licenses under which the source code is released. A codebase history shows the evolution of the source code of a project. It specifically shows the total size of a project's source code over time. This graph reveals at a glance how long the project has been around, and the relative pace of development over time. It's generally a good sign to see sustained, constant activity over a long period of time. This means that people are continually updating it (fixing bugs and/or improving features), and that the project has staying power. The graph of Figure 6 for instance was generated by the Ohloh.net site.

Towards Tool Support Such use of publicly available data is also made by different tools for analyzing the F/OSS development process. For instance, CVSAnalY [50] is a tool that extracts and manages statistical information out of the activity that happens in a control version repository such as CVS and most recently Subversion. It parses repository logs and transforms them in either information exchange XML and database SQL formats. It has a web interface - called CVSAnalYweb - where the results can be retrieved and analyzed in an easy way. Another example of such an analysis tool is DrJones [49]. This software supports a software archeology analysis on software that is stored in a

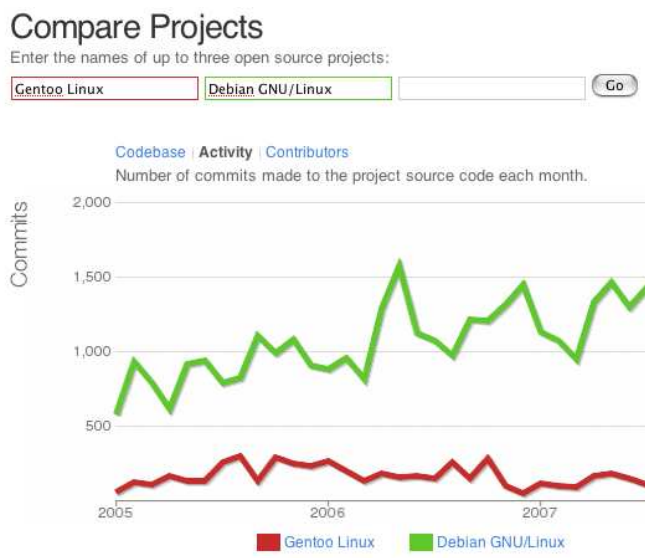

Figure 6: Ohloh comparison of Gentoo and Debian Linux activity (February 22nd, 2008)

CVS or Subversion versioning repository. DrJones analyzes how old the software system is on a per-line basis and extracts figures and indexes that make it possible to identify how 'old' the software is, how much it has been maintained and how much effort it may take to maintain it in the future. Dr. Jones counts the SLOC, the number of files, the number of authors and gathers file timestamps to evaluate occurred changes. A list of other tools used to extract information about the F/OSS development process are listed and made available on the Libresoft website [63].

4.3 F/OSS Process Management

While many project managers and developers are already familiar, thanks to their experience or to the literature [119], with the importance of F/OSS processes, there is no consensus on what a F/OSS process is composed of, nor how it should be implemented in some project environment.

4.3.1 General Process Management

Numerous process-related standards are available for defining and executing processes. A process in the field of business information systems is a set of activities that aim to achieve some (business) objective. Activities can involve people interaction, and several activities may have to be coordinated – or *orchestrated* – for the business objective to be met. Existing work on this topic can be divided into work on **what** needs to be expressed, **how** activities are expressed, and an examination of accrued **benefits**.

On the **what** of process modeling, an overview of how F/OSS projects are organized is presented in [28]. The paper explains related terminology and

overviews key processes. The processes include decision-making within the project management, accountability of bugs to packages, communication among developers, generation of awareness about the project in the software community, managing source code, testing and release management.

On the **how** of process modeling, the Business Process Modeling Language (BPML) [11] is a meta-language for the modeling of business processes, like XML is a meta-language for the modeling of business data. BPML provides an abstracted execution model for collaborative and transactional business processes. It has been published by Business Process Management Initiative (BPMI) [10] as a standard providing a general approach to express business processes in organizations. BPML is a rival language with other standards such as IBM's WSFL (Web Services Flow Language) [62] and Microsoft's XLANG [116] (Web Services for Business Process Design).

JBoss jBPM [57] is a platform for multiple process languages supporting workflow, BPM, and process orchestration. JBPM enables the creation of business processes that coordinate people, applications and services. The JBoss jBPM process designer graphically represents the business process steps to facilitate a strong link between the business analyst and the technical developer, e.g., Figure 7. Other related systems include ObjectWeb Bonita [9], Apache Agila [4] and the Apache Open For Business Project [7].

In [103] the authors highlight issues in the modeling of techno-social processes found in F/OSS development. They focus on the modeling of Apache Web server and Mozilla browser project processes. As previously existing descriptions of these processes are informal and narrative, they allow no analysis, visualization, computational enactment, reuse or comparison. The authors use the *rich pictures* [77] method for discovering F/OSS processes to organize, associate observed development roles, tools and tasks. To provide a way to understand and perform these processes, the use of PML (Process Modeling Language) [82] is proposed.

Among the **benefits** of modeling Open Source processes, the authors in [103] cite the help to new contributors for the enactment of processes. Other benefits include the enabling of continuous process improvement techniques, the provision of a coordination resource that can be used by distributed developers to synchronize their activities, roles and artifacts. Recent progress in the development of automated mechanisms to support and streamline the process discovery effort has been then described by the authors in [59].

The impact of software process maturity on Free Software project success has been studied in [73] through statistical analysis of 40 successful and 40 unsuccessful, randomly chosen, projects. The results show that the maturity of some processes are linked to the success of a project. This study identified the importance of the use of version control tools, effective communication through the deployment of mailing lists, and found several effective strategies related to testing. The identification of processes employed by successful free software projects is of substantial value to practitioners since they give an indication of the areas deserving attention.

The EU EDOS Project (Environment for the development and Distribution of Open Source software) [1, 93] developed a Process Reference Model (PRM) to describe all artifacts of the F/OSS process. The goal of the PRM is to define the key content and community artifacts of F/OSS and to formalize the relations between these. The model allows integrity rules to be defined on all artifacts,

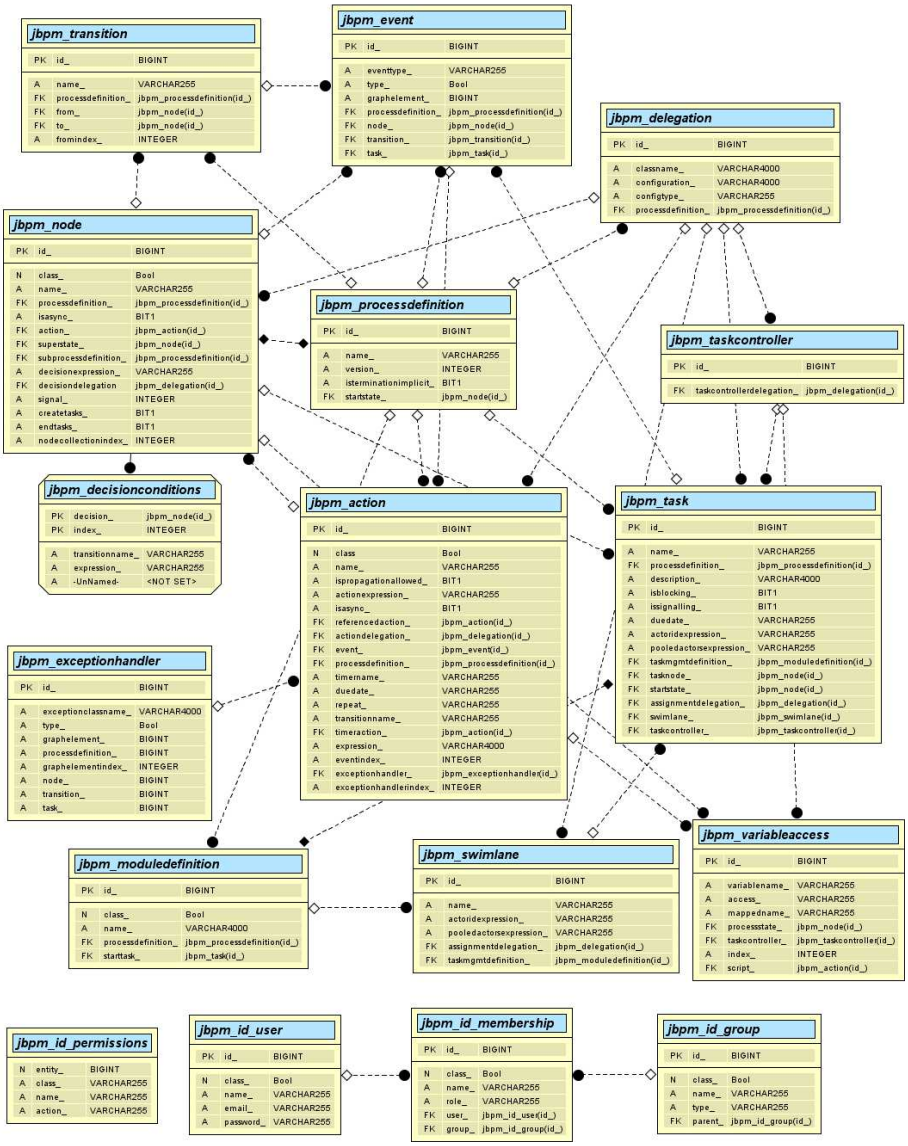

Figure 7: Process Description from JBoss

e.g., that packages must have licenses or that developers cannot be testers for the same package. The PRM can be used in several ways. For instance, it can serve as a basis for comparing the processes used by various F/OSS distributor. It can also serve as the design of an information system for a new project. This information system would act as a real-time dashboard for the project and ensure that the integrity rules are respected for all F/OSS operations.

Perhaps one of the most important process efforts of recent times is OpenUP – the Eclipse Process Framework. This is built over OMG's SPEM (Software Process Engineering Meta-model, version 2). The meta-model expresses core engineering artifacts such as tools, white-papers, guidelines, estimates, etc. The importance of OpenUP is that the model is supported by the Eclipse community, and is thus a major step in the standardization of process modeling in the open source domain. OpenUP does not model all aspects required for F/OSS processes, notably community profiling and project interoperability. However, the OpenUP tools are designed to be customizable, so it may be possible to build community management elements over OpenUP.

4.3.2 Support for Distribution and Production Processes

The goal of these tools is to aid F/OSS production processes. They range from basic tools like desktop development environments, like Eclipse, and build tools like Apache Ant [5], to wholesale collaborative environments like Maven.

Maven [6] is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central project description. Maven was originally started as an attempt to simplify the build processes in the Jakarta Turbine project [8]. There were several projects each with their own Ant build files, that were all slightly different, and whose JARs were checked into CVS. The motivation for Maven was a standard way to build the projects, a clear definition of published project information and a mechanism for JAR sharing across projects. There are several areas of concern that Maven addresses: making the build process easy, providing a uniform build system, providing quality project information, providing guidelines for best practices development and allowing transparent migration to new features.

Amos [16], is EU funded project that aimed at making it easier to build software based on the composition of Open Source Code. One of the difficulties in this case is finding the right pieces of code, a task that usually requires deep knowledge of many software products. The idea behind Amos is to use high level descriptions of code assets, and perform a search using these descriptions to find the set of packages which best (according to some measure) fulfills user requirements. The idea has been materialized in a tool, composed of a user interface, a database of descriptions, and a matching engine that interacts both with the user (through the interface) and with the database where descriptions are stored.

QSOS is a method for qualifying, comparing and selecting F/OSS in an objective, traceable and argued way [95]. It relies on interdependent and iterative steps aimed at generating software ID cards and evaluation sheets that support the selection of the best solution in a given context. This method defines frames of reference based on licenses, communities, functional grids that help evaluating software in terms of functional coverage and risk for both users and

service providers. The selection and comparison of software is then done in the scope of an evaluation context. QSOS aims at improving software description and facilitating selection.

The OpenSuse *build service* [91] provides a complete distribution development platform to create Linux distributions based on SUSE Linux. Its open interfaces allow external services to interact with the build service and use its resources. A server infrastructure hosts all sources, provides a build system to create packages, provides a download and mirror infrastructure for distributing packages and serves as the communication framework. Interaction with the build service can be done through an open API, a web interface or via a command line. The OpenSuse service aims at connecting open source communities, providing a means to develop distributions while controlling issues like dependencies.

Red Hat Network (RHN) [124] is an architecture that essentially focuses on code distribution. Indeed, RHN is used to download distribution ISOs, patches and software packages as well as to update systems based on user customization. The network is accessible through an *Access API*. The key abstraction RHN provides is the notion of *channels*, which corresponds to a set of packages. Every client machine that is connected to a specific channel can be updated when the content of the channel changes. A base channel corresponds to the core system and other types of channels are built on top of it. For instance, developers to distribute their work use a development channel. A Testing & QA channel is used for bug reporting. The architecture defines actions for each channel. An example action could be to remove packages whenever a new version is available, or to rollback to a previous version of the system when a compilation error occurs.

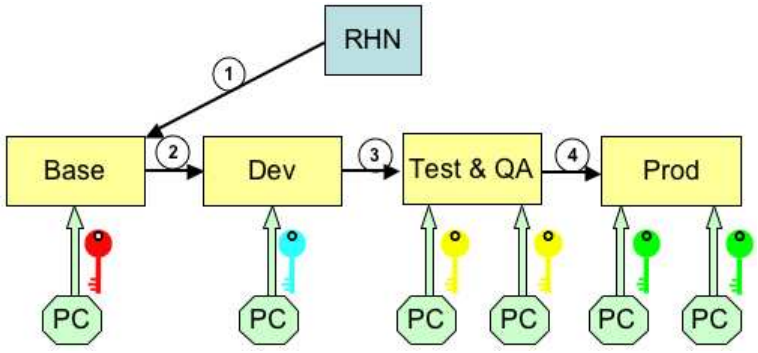

Figure 8: Red Hat Network API

4.3.3 F/oss Interoperability

Interoperability is the ability for a system or a product to work with other systems or products without special effort on the part of the user. Increasingly, enterprises are cooperating with other enterprises and competitiveness is largely determined by the ability to seamlessly interoperate with others. As such, F/OSS process improvement is directly bound to the ability of projects to interoperate.

Thus this issue also needs to be considered in the F/OSS context, especially given the multitude of F/OSS projects in existence.

Some projects like SourceForge [107] or GForge [48] offer a whole development infrastructure for F/OSS development that provides a meeting point for producers and users where they can exchange information. They classify projects and propose tools to browse projects by properties such as development status, intended audience, license, translations, etc. However, existing services only provide information about the projects using them. The freshmeat [45] website tries to tackle some of these issues by providing a central index for F/OSS projects enabling them to disseminate information about themselves, and the freshports [46] website lets people browse the entire FreeBSD [43] ports collection, it provides cross references, charts, graphs and links. However, these solutions only allow sparse interaction between projects and potential users.

The Flink group [31] has defined a Linux ontology and aims at demonstrating the benefits that result from formalizing knowledge about the Linux operating system. In [52], the authors discuss the application of ontology-based knowledge engineering to Linux. Various possible applications such as package management or information search are discussed which would all benefit from a comprehensive ontology of the domain. The use of ontology to describe Linux provides a common ground of understanding.

The Friend of a Friend (FOAF) [34] project is creating a Web of machine-readable pages describing people, the links between them and the things they create and do. It is a community driven effort to define an RDF vocabulary for expressing metadata about people, and their interests, relationships and activities. FOAF is tackling head-on the wider Semantic Web goal of creating a machine processable web of data. It facilitates the creation of the Semantic Web equivalent of the archetypal personal homepage: My name is Leigh, this is a picture of me, I'm interested in XML, and here are some links to my friends. Just like the HTML version, FOAF documents can be linked together to form a web of data, with well-defined semantics.

DOAP [24] is a project to create an XML/RDF vocabulary to describe open source projects. This project aims at providing an internationalizable description of a software project and its associated resources, including participants and Web resources as RDF schemas. The project provides basic tools to enable the easy creation and consumption of such descriptions in all the popular programming languages and ensures interoperability with other popular Web metadata projects (RSS, FOAF, Dublin Core). Finally, DOAP vocabulary is extensible for specialist purposes. Note that DOAP does not aim at handling software releases, or at planning data internal to the project such as task assignments or milestones.

QualiPSo [96] is a European IP project working on open source process improvement. It's goal is to investigate and implement development processes through an open forge, including business models, methods and tools to foster the wide adoption of Open Source Software by European organizations from ITC players to end users. It focuses on aspects such as license management, documentation and information management, and interoperability. Unlike other projects focusing on the technical or semantic aspect of interoperability, QualiPSo aims at handling all the aspects of interoperability mentioned above. It will provide a model for information contained in F/OSS including elements such as project, participants, tasks and planning, requirements, bugs,

documents, new functionality proposals, source code, project versions, mail, forums, meetings, source code management.

5 Conclusion: A fragmented World

As seen, organization, process management, interoperability and production methods are hot topics and they all contribute to process improvement. Nonetheless, no existing approach or tool provides consistent support for complete F/OSS process support, e.g., to conduct streamlining. The second major weakness of current approaches is that community management is generally weak; no system achieves the type of functionality requirements that we outline for community management in Section 3.

The same weaknesses exist at the level of tool support. Tools generally address specific activities of the F/OSS process, without being integrated in a large picture. Examples include defect management tools (Bugzilla [13]), testing tools (Bugzilla test runner [14], Fitnesse [30], Salome [101]) and other tools (SourceForge) [107], GForge [48], Alioth [3], Libresource [64]) providing an integrated solutions for managing F/OSS projects. The dashboards provided by some projects [26, 44] exemplify this issue as they focus on specific information that cannot be reused across projects.

We have seen that F/OSS tools for process and workflow management exist, however these tools are often designed for the enterprise environment, and do not consider the specific requirements of the F/OSS environment. Further these tools imply a machinery which while being mandatory in the enterprise context and very useful in both enterprise and F/OSS contexts, might be difficult to implement in an open environment.

In such a context we can argue that the issue of F/OSS process improvement is left open. A transversal approach to the process management issue is required. It has to offer means to handle project as a whole considering involved activities, their content, community, and allowing reasoning about F/OSS processes, tasks, metrics, etc. in a distributed context in order to enable F/OSS process streamlining. On the bright side, the technological bricks of the solution exist in the public domain and through the efforts of numerous research and development projects.

References

[1] S. Abiteboul, R. D. Cosmo, S. Fermigier, S. Lauriere, and al. Edos: Environment for the development and distribution of open source software. In *Proceedings of the First International Conference on Open Source Systems, Genova, Italy*, July 2005.

[2] S. Abiteboul, I. Dar, R. Pop, G. Vasile, D. Vodislav, and N. Preda. Large scale P2P distribution of open-source software. In C. Koch, J. Gehrke, M. N. Garofalakis, D. Srivastava, K. Aberer, A. Deshpande, D. Florescu, C. Y. Chan, V. Ganti, C.-C. Kanne, W. Klas, and E. J. Neuhold, editors, *VLDB*, pages 1390–1393. ACM, 2007.

[3] Alioth. `http://alioth.debian.org/`, June 2006.

[4] Apache agila. http://wiki.apache.org/agila/, Jan. 2007.

[5] Apache ant: build tool. http://ant.apache.org/, October 2006.

[6] Apache Project. Apache maven: software project management and comprehension tool. http://maven.apache.org/, October 2006.

[7] Apache Project. Apache open for business (OFBiz). http://ofbiz.apache.org/, October 2006.

[8] Apache Project. Jakarta turbine project. http://jakarta.apache.org/turbine/, Jan. 2007.

[9] Objectweb bonita. http://wiki.bonita.objectweb.org/xwiki/bin/view/Main/WebHome, Jan. 2007.

[10] Business process management initiative (BPMI). http://www.bpmi.org/, Dec. 2006.

[11] Business process management language 1.0 (BPML). http://www.ebpml.org/bpml_1_0_june_02.htm, Jan. 2007.

[12] Business readiness rating - a framework for evaluating open source software. http://www.openbrr.org/wiki/index.php/Home, Jan. 2007.

[13] Bugzilla. http://www.bugzilla.org/, June 2006.

[14] Bugzilla test runner. http://www.willowriver.net/products/testrunner.php, June 2006.

[15] Co-ordination action for libre software engineering for open development platforms for software and services (CALIBRE) project. http://www.calibre.ie/, September 2006.

[16] M. Carro. The amos project: An approach to reusing open source code. In *Proceedings of the CBD 2002 / ITCLS 2002 CoLogNet Joint Workshop*, pages 59–70. Facultad de Informatica, September 2002.

[17] B. Cohen. Incentives Build Robustness in BitTorrent. In *Proceedings of the Workshop on Economics of Peer-to-Peer Systems*, Berkeley, CA, USA, 2003.

[18] Compiere. http://www.compiere.org/product/index.html, October 2006.

[19] Common public license (CPL) - v1.0. http://www.eclipse.org/legal/cplv10.html, September 2006.

[20] Concurrent versions system (CVS). http://www.nongnu.org/cvs/, October 2006.

[21] Debian Project. Debian developer locations. http://www.debian.org/devel/developers.loc, October 2006.

[22] Debian Project. Debian new maintainer's guide. http://www.debian.org/doc/maint-guide/, October 2006.

[23] Debian Project. Debian social contract. http://www.debian.org/social_contract, October 2006.

[24] Description of a project (DOAP). `http://usefulinc.com/doap/`, Jan. 2007.

[25] Eclipse project. `http://www.eclipse.org/`, September 2006.

[26] Eclipse Project. Eclipse project dashboards. `http://www.eclipse.org/projects/dashboard/`, June 2006.

[27] EDOS WP2 Team. Deliverable d2.1.: Report on Software Management Dependencies. Technical report, EDOS Project, 2005.

[28] J. Erenkrantz and R. Taylor. Supporting distributed and decentralized projects: Drawing lessonsfrom the open source community. In *Proceedings of 1st Workshop on Open Source in an Industrial Context, Anaheim, California.*, October 2003.

[29] Exim internet mailer. `http://www.exim.org/`, Dec. 2006.

[30] Fitnesse. `http://www.fitnesse.org`, June 2006.

[31] Formalized linux knowledge (Flink). `http://flink.dcc.ufba.br/en/`, June 2006.

[32] FLOSSMetrics. Free/Libre Open Source Software Metrics (FLOSSMetrics) Project. `http://www.flossmetrics.org/`, Jan. 2007.

[33] FLOSSMole. Free/Libre Open Source Software Mole (FLOSSMole). `http://ossmole.sourceforge.net/`, Feb. 2007.

[34] Friend of a friend (FOAF). `http://www.foaf-project.org/`, Jan. 2007.

[35] C. W. Frans-Willem Duijnhouwer. Open source maturity model (OMMM). Technical report, Capgemini, 2003.

[36] Free Software Foundation. Gnu is not unix (GNU) operating system. `http://www.gnu.org/`.

[37] Free Software Foundation. Overview of the gnu project. `http://www.gnu.ai.mit.edu/gnu/gnu-history.html`, December 1998.

[38] Free Software Foundation. The free software definition. `http://www.gnu.org/philosophy/free-sw.html`, October 2006.

[39] Free software foundation (FSF). `http://www.fsf.org/`, October 2006.

[40] Free Software Foundation. Free software philosophy. `http://www.gnu.org/philosophy/`, October 2006.

[41] Free Software Foundation. Free software: Various licenses and comments about them. `http://www.gnu.org/licenses/license-list.html`, October 2006.

[42] Free Software Foundation. Gnu general public license. `http://www.gnu.org/copyleft/gpl.html`, October 2006.

[43] Freebsd project. `http://www.freebsd.org/`, September 2006.

[44] FreeBSD Project. Freebsd dashboards. `http://people.freebsd.org/~bsd/prstats/`, June 2006.

[45] freshmeat.net. `http://www.freshmeat.net/`, October 2006.

[46] Freshports - the place for ports. `http://www.freshports.org/`, October 2006.

[47] Gaim linux/unix instant messenger client. `http://gaim.sourceforge.net/`, October 2006.

[48] Gforge. `http://gforge.org/`, June 2006.

[49] J. M. G.-B. Gregorio Robles and I. Herraiz. An empirical approach to software archeology. In *21st IEEE International Conference On Software Maintenance*, Sept. 2005.

[50] S. K. Gregorio Robles and J. M. Gonzlez-Barahona. Remote analysis and measurement of libre software systems by means of the cvsanaly tool. In *Proceedings of the 2nd ICSE Workshop on Remote Analysis and Measurement of Software Systems (RAMSS '04). 26th International Conference on Software Engineering (Edinburgh, Scotland)*, May 2004.

[51] J. C.-G. V. M.-O. Gregorio Robles-Martnez, Jess M. Gonzlez-Barahona and L. Rodero-Merino. Studying the evolution of libre software projects using publicly available data. In *Proceedings of the 3rd Workshop on Open Source Software Engineering at the 25th International Conference on Software Engineering*, May 2003.

[52] D. A. Guillaume Barreau. Semantic linux: a fertile ground for the semantic web, Apr. 2005.

[53] Idabc open source observatory. `http://europa.eu.int/idabc/en/chapter/452/`, September 2006.

[54] ISO. 8402:1994- Quality management and quality assurance – Vocabulary. `http://www.iso.org/iso/en/`, Dec. 2000.

[55] ISO/IEC. 9646-7:1995 - Information technology – Open Systems Interconnection – Conformance testing methodology and framework. `http://www.iso.org/iso/en/`, June 2001.

[56] K. C. James Howison, Megan Conklin. Ossmole: A collaborative repository for floss research data and analyses. In *Proceedings of the First International Conference on Open SourceSystems, Genova, Italy*, July 2005.

[57] Jboss jbpm. `http://www.jboss.com/products/jbpm/`, Jan. 2007.

[58] N. J. Jean-Michel Dalle. Open-source vs. proprietary software. Jan. 2002.

[59] C. Jensen and W. Scacchi. Automating the discovery and modeling of open source software development process. In *3rd. Workshop on Open Source Software Engineering, 25th. Intern. Conf. Software Engineering, Portland, OR*, may 2003.

[60] Project jxta. `http://www.jxta.org/`, September 2006.

[61] A. M. S. Laurent. *Understanding Open Source and Free Software Licensing.* O'Reily & Associates, Inc., 1 edition, August 2004.

[62] F. Leymann. Web services flow language (WSFL 1.0). Technical report, IBM Software Group, 2001.

[63] Libresoft/GSyC. Libre software engineering tools. `http://libresoft.urjc.es/Tools/index_html`, Mar 2007.

[64] Libresource. `http://dev.libresource.org/`, June 2006.

[65] Linux desktop testing project. `http://ldtp.freedesktop.org/wiki/`, Jan. 2007.

[66] Linux test project. `http://ltp.sourceforge.net/`, Jan. 2007.

[67] Malaysian public sector open source software initiative. `http://opensource.mampu.gov.my`, September 2006.

[68] Mandriva. Mandriva club. `http://club.mandriva.com/`, Dec. 2006.

[69] Mandriva. Mandriva Linux package statistics. http://mandriva.edos-project.org/xwiki/bin/view/Packages/PackageStatistics, Oct 2006.

[70] Mandriva. Mandriva quality assurance lab's contributor's corner. `http://qa.mandriva.com/twiki/bin/view/Main/QaContributorsCorner`, Jan. 2007.

[71] S. M. Maurer, A. Rai, and A. Sali. Finding cures for tropical diseases: Is open source an answer? *PLoS Med*, 1(3), December 2004.

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

[73] M. Michlmayr. Software process maturity and the success of free software projects. In K. Zieliski and T. Szmuc, editors, *Software Engineering: Evolution and Emerging Technologies*, pages 3–14, Krakw, Poland, 2005. IOS Press.

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

[75] M. Michlmayr and A. Senyard. A statistical analysis of defects in debian and strategies for improving quality in free software projects. In J. Bitzer and P. J. H. Schrder, editors, *The Economics of Open Source Software Development*, pages 131–148, Amsterdam, The Netherlands, 2006. Elsevier.

[76] Microsoft. Microsoft shared source initiative. `http://www.microsoft.com/resources/sharedsource/`, Jan. 2007.

[77] A. Monk and S. Howard. The rich picture: A tool for reasoning about work context. March-April 1998.

[78] Mozilla project. `http://www.mozilla.org/`, September 2006.

[79] B. Munos. Can open-source r&d reinvigorate drug research? Technical report, Eli Lilly & Co., September 2006.

[80] Navica Inc. The open source maturity model (OSMM). `http://www.navicasoft.com/pages/osmmoverview.htm`, Jan. 2007.

[81] Neogia. `http://neogia.labs.libre-entreprise.org/index.html`, October 2006.

[82] J. Noll and W. Scacchi. Specifying process-oriented hypertext for organizational computing. *Network and Computer Application*, 24(1):39–61, 2001.

[83] Ohloh.net. Ohloh.net Project. `http://www.ohloh.net/`, Jan. 2007.

[84] Open Source Initiative. Approved licenses. `http://www.opensource.org/licenses/`, October 2006.

[85] Open Source Initiative. Open source definition. `http://www.opensource.org/docs/definition.php`, October 2006.

[86] Open source initiative (OSI). `http://www.opensource.org/`, October 2006.

[87] OpenBRR. Business readiness rating for open source - brr 2005 rfc 1. Technical report, www.openbrr.org, 2005.

[88] Openoffice.org. `http://www.openoffice.org/`, September 2006.

[89] OpenOffice.org. Open office automated gui testing project. `http://qa.openoffice.org/qatesttool/`, Jan. 2007.

[90] Opensourcetesting.org. `http://www.opensourcetesting.org/`, Jan. 2007.

[91] OpenSuse. Opensuse build service. `http://en.opensuse.org/Build_Service`, June 2006.

[92] Opentaps. `http://www.sequoiaerp.org`, October 2006.

[93] M. Pawlak and C. Bryce. A reference model for f/oss project management. In *FOSDEM*, 2007.

[94] B. Perens. The open source definition. In *Open Sources: Voices from the Open Source Revolution*. O'Reily & Associates, Inc., 1 edition, January 1999.

[95] Method for qualification and selection of open source software (QSOS) project. `http://www.qsos.org/`, June 2006.

[96] Quality platform for open source software (QualiPSo) project. `http://www.objectweb.org/phorum/download.php/16,271/QualiPSo_PM_Oct4.pdf`, Jan. 2007.

[97] E. S. Raymond. The cathedral and the bazaar. `http://www.openresources.com/documents/cathedral-bazaar/`, Aug. 1998.

[98] G. Robles, J. M. Gonzalez-Barahona, and M. Michlmayr. Evolution of volunteer participation in libre software projects: Evidence from Debian. In M. Scotto and G. Succi, editors, *Proceedings of the First International Conference on Open Source Systems*, pages 100–107, Genova, Italy, 2005.

[99] G. Robles, J. M. Gonzalez-Barahona, M. Michlmayr, and J. J. Amor. Mining large software compilations over time: Another perspective of software evolution. In *Proceedings of the International Workshop on Mining Software Repositories (MSR 2006)*, Shanghai, China, 2006.

[100] Rth. `http://rth-is-quality.com/`, Jan. 2007.

[101] Salome test management tool. `https://wiki.objectweb.org/salome-tmf/`, June 2006.

[102] R. S. Sandhu. Role-based access control. *Advances in Computers*, 46:238–287, 1998.

[103] W. Scacchi. Issues and experiences in modeling open source software processes. In *3rd. Workshop on Open Source Software Engineering, 25th. Intern. Conf. Software Engineering, Portland, OR*, may 2003.

[104] Science commons: Accelerating the scientific research cycle. `http://sciencecommons.org/`, November 2006.

[105] Science, education and learning in freedom (SELF) project. `http://www.selfproject.eu/`, Jan. 2007.

[106] Sendmail smtp server. `http://www.sendmail.org/`, Dec. 2006.

[107] Sourceforge. `http://sourceforge.net/`, June 2006.

[108] Sourcetap. `http://sourcetapcrm.sourceforge.net`, October 2006.

[109] Software testing automation framework (STAF). `http://staf.sourceforge.net/`, Jan. 2007.

[110] Sun Microsystems Inc. Openoffice.org testtool: Introduction to automated gui testing. Technical report, OpenOffice.org, Sept. 2006.

[111] Sun Microsystems Inc. Sun community source licensing (SCSL). `http://www.sun.com/software/communitysource/`, Jan. 2007.

[112] Test case web (TCW). `http://sourceforge.net/projects/tcw`, Jan. 2007.

[113] Tightening knowledge sharing in distributed software communities by applying semantic technologies (TEAM) project. `http://www.team-project.eu/`, Jan. 2007.

[114] Testlink. `http://testlink.sourceforge.net/`, Jan. 2007.

[115] Testopia test case management. `www.mozilla.org/projects/testopia/`, Jan. 2007.

[116] S. Thatte. Web services for business process design. Technical report, Microsoft Corporation, 2001.

[117] L. Torvalds. The story of the linux kernel. In S. O. Chris DiBona, Mark Stone, editor, *OpenSources: Voices from the Open Source Revolution*. O'Reilly an Associates, February 1999.

[118] Tropical disease initiative. `http://www.tropicaldisease.org/`, November 2006.

[119] D. Tuma. Open source software: Opportunities and challenges. *STSC Crosstalk The Journal of Defense Software Engineering*, 18(1):6–10, Jan. 2005.

[120] D. Weiss. Quantitative analysis of open source projects on sourceforge. In *Proceedings of the First International Conference on Open SourceSystems, Genova, Italy*, July 2005.

[121] Wikipedia, the free encyclopedia. `http://www.wikipedia.org/`, Nov. 2006.

[122] Wikipedia. Shared source on wikipedia. `http://en.wikipedia.org/wiki/Shared_source`, Jan. 2007.

[123] Wiktionary, the free dictionary. `http://www.wikionary.org/`, Nov. 2006.

[124] S. Witty. Best Practices for Deploying and Managing Linux with Red Hat Network, Dec. 2004.

[125] Xplanet. `http://xplanet.sourceforge.net/`, October 2006.

Centre de recherche INRIA Rennes – Bretagne Atlantique
IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399