Towards an Openness Rating System for Open Source Software

Wolfgang Bein
Center for the Advanced Study of Algorithms
School of Computer Science
University of Nevada
Las Vegas, NV 89154
bein@cs.unlv.edu

Clinton Jeffery
Department of Computer Science
University of Idaho
Moscow, ID 83844-1010
jeffery@uidaho.edu

Abstract

Many open source software projects are not very open to third party developers. The point of open source is to enable anyone to fix bugs or add desired capabilities without holding them hostage to the original developers. This principle is important because an open source project's developers may be unresponsive or unable to meet third party needs, even if funding support for requested improvements is offered.

This paper presents a simple rating system for evaluating the openness of software distributions. The rating system considers factors such as platform portability, documentation, licensing, and contribution policy. Several popular open source products are rated in order to illustrate the efficacy of the rating system.

1 Motivation, Background and Related Work

The spirit of open source software can perhaps be illustrated by how mathematical knowledge is made available to society. A recent volume by the combinatorialists Aigner and Ziegler "Proofs from The Book" [1] gives a number of theorems, which are chosen because they have beautiful and ingenious proofs. In fact in that volume, for most theorems a number of different proofs are given, though many theorems carry the name of one famous mathematician. One such example is Turán's graph theorem, which states that when a graph $G = (V, E)$ on n vertices has no p-clique [1] ($p \geq 2$) then for the number of edges $|E|$, the upper bound $|E| \leq (1 - \frac{1}{p-1})\frac{n^2}{2}$ holds. The volume gives the original Turán proof, as well as four alternative proofs. The theorem roughly says that if a graph has very many edges then there necessarily have to exist large cliques in the graph. Knowing this is useful in the reliability analysis of communication networks, for example.

If mathematics followed the practices of proprietary software then anyone interested in the aforementioned theorem would be able to see the statement of the theorem only by paying a fee (say, by purchasing this article from a publishing house), but the inner workings of Turán's proof would not be accessible for any price. Worse, anyone publishing any of the alternative proofs might have to fear being sued by Turán's estate. Of course, this would be complete folly, and indeed in the real world of mathematics anyone is free to prove Turán's as they please. Yet, it is still possible to engage in profitable enterprise, as Springer Verlag did with their best-selling volume (despite the fact that – in principle – Turán's theorem can be found on the world wide web.)

Imagine that a modern day Turán felt compelled to keep the proof of his theorem under wraps. As an upstanding mathematician, he really does not have the option to state his discovery (the theorem) on his website, while at the same time completely withholding the proof. A reason for this modern Turán to have his method not come out too soon could be a desire to secure government funding before anyone else. So in order to bask in the glory of his result without giving away everything, he would publish a pretend-proof, a proof not altogether false but instead completely incomprehensible. This way Turán can claim to follow the ways of the scientific community while still obstructing true open access.

This is very similar to practices around many so-called open source projects. Recently, a colleague of ours undertook to extend a popular piece of open

[1] A subset of vertices of cardinality p is said to be a p-clique if for every two vertices in the subset, there exists an edge connecting the two.

IEEE computer society

source software, only to have her initial efforts stymied. The software was developed largely with public funds, but seemed to have adopted open source status as a political ploy. The source code was available, but without comments or build instructions. Naively handing it off to its compiler, one got various dependency errors and a failed build. The frustration was documented by several persons on the project web forums, with the developers in complete denial that there was any problem with (almost) no one in the world able to build their "open source" software but them.

2 Cathedral and Bazaar

In the preceding example, the software's website stated that the code was available, but code submissions were unwelcome. This is a separate issue; the superiority of the cathedral model of software development over the bazaar is a philosophical question where different reasonable persons may disagree and still produce excellent open source software [4]. Corporate labs and elite schools share a fondness of and desire to control their open source projects. In modern days when a corporation open sources a proprietary commercial system, such as SGI's open sourcing GL or Sun's OpenSolaris or OpenJDK, it is easy to argue that the cathedral-control is economically motivated [2]. However, when Ralph Griswold went from AT&T to a university and started giving his product away by literally placing it in the public domain, he still adhered to a cathedral model as the only way to maintain quality in an otherwise anarchic academic schedule. The cathedral model did not preclude vigorous efforts to make the software as usable by others as possible. Many other major academic research projects that manage to produce a software release are similar; the projects' university budgets do not include time to support a bazaar. The cathedral does not always mean user-written contributions are unwelcome, it just defines the terms of engagement.

Beyond Raymond's famous essay, Nathan Willis' article on linux.com [6] describes explicitly closed development processes for open source software which contradict the nature of their licenses, with examples including KDE's Oxygen and the GIMP's UI design.

While these examples are ironic and cathedral-style open source development may be frowned upon by some, they do not violate the main founding principle of open source, which is that users are free to adapt, maintain, and fix the software as needed. It is one thing to use the cathedral model and maintain control over one's own development, and it is another to pose as an open source project while minimizing the ability of others to compile the code.

3 The Social Contract of Open Source

In our own open source software efforts, we were aware of how much work it is to try and make a working program buildable by others: it remains an ongoing challenge. However, from the example described earlier, no comments or build instructions in software developed by one of the leading software engineering institutions in the world? It seemed deliberate.

We began to wonder how many other software projects were buildable only by the developers who wrote them. We have seen many open source projects that did not compile readily for reasonably proficient technical persons. When this is due to a lack of budget or developers' technical abilities, it is understandable and requires no further comment. However, among larger projects that ought to be buildable, we quickly concluded that openness was not a Boolean variable, but a complex and continuous space. Different open source software projects range wildly in terms of how friendly or hostile they are to developers.

We view the violation of the social contract inherent in the "closed" open source project from the point of view of the scientific method. All open source software projects are science experiments; they have to be reproducible (in the case of software, buildable and runnable) by others in order for the results (say, the claim that the source code release was complete and correct) to be confirmed.

Daniel E. Markle's blog (dated Thursday December 8 2005) contains an article titled Community and "Fake" Open Source Projects [3] that lists four ways to kill open source software: inadequate user documentation, inadequate developer documentation, no open bug tracking system, and no access to the development work in progress. We view the first two of these as essential elements and include them in our ratings system in this paper, while the latter two are basic to a successful bazaar, and highly desirable, but not intrinsic to whether the end user is able to use the software for their own purposes.

Most software ratings systems will evaluate quality, performance, or suitability for a given constituency's needs. For example, the Business Readiness Ratings (BRR) system (see `www.openbrr.org`) defines twelve categories to evaluate open source software: usability, scalability, and so on. (See also [5].) These are largely classic software engineering quality metrics. The ratings system presented in this paper is focused on the single issue of openness.

4 The Openness Factor Rating Scheme

This section presents the categories and a rubric used to assess openness of different open source software. We expect the rubric deserves correction and expansion and solicit input on it. Our Openness Factor Rating (OFR) takes the geometric average of ratings for n categories $C_1, C_2, \ldots, C_n$, taking values between 0 and 1:

$$\text{OFR} = (C_1 \cdot C_2 \cdot \ldots \cdot C_n)^{\frac{1}{n}}$$

For our purposes we consider exactly nine categories (ie., $n = 9$) and we describe below how to obtain ratings $C_1, C_2, \ldots, C_n$. We note that the different openness categories are not smooth linear distributions: some categories are heavily skewed towards the value 1.0, as all open source projects are near each other in that category.

Arithmetic mean gives less useful numbers in our opinion. We also note that one could simply take the minimum score in any one component, but that equates single category offenders with multi-category offenders. The validity of the current metric is not asserted. Rather, the purpose of the paper is to generate discussion and development of a metric that would be more or less widely acceptable. However, in Section 5 we give a number of examples, which show the usefulness of our measure.

Language portability Not all open source code is written in a language that runs everywhere. Unfortunately, even code written in a portable language often will not compile using other compilers besides the one it was developed on. This includes language version dependence, as in: code only compiles under Java 1.5.x, do not use 1.4.x or 1.6.x.

Rubric: a starting score of 1.0 is assigned for languages C, C++, Java. Other HLL's get a factor equal to the market share of platforms on which they can be built; min(.8, sum(.1 per mainstream CPU architecture for assembler)).

Library portability The more libraries required and the more version dependencies, the more difficult it is for independent third parties to build and use the software. Some libraries are more universal than others; if they are part of a language standard that has survived multiple decades, they are not much of an obstacle to openness.

Rubric: A score of 1.0 for software with no external libraries that are not part of an ANSI or ISO language standard. Factor the market shares of all required libraries not provided with the source distribution. Apply penalties for libraries without binary distributions, libraries available for only some compilers, and libraries larger and more complicated than the application that uses them (e.g. Boost is often larger than the application that uses it, and tricky to get working for particular compilers).

Contributors The more people who have contributed to the software, the more convincing the argument is that others have been able to build and run the software in the past.

Rubric: A score of 1.0 = 1,000 contributors. .9 = 100 contributors, .8 = 10+ contributors.

Users The more users a piece of software has, the more eyeballs have seen its text messages, and the more international and multi-cultural it is likely to be.

Rubric: A score of 1.0 = 100,000 users; .9 = 10,000 users; .8 = 1000 users; .7 = 100 users

Build Documentation Build documentation would include README files, install.txt files, Makefiles, Ant files, and so on. They affect openness in a fundamental way.

Rubric: A score of 1.0 = there are technical reports (or multi-page documents) on how to build the system, building is straightforward; .9 = there are environment variables that must be set, or a non-standard configuration step; .8 = the build is documented but has multiple manual steps requiring reading; .7 = the build requires study (say, a half hour) of build documentation; .5 = the build requires training, or a day or more to work through; 0.1 = there is no build documentation.

User Documentation User documentation is a major component of making a piece of open source software usable by others.

Rubric: A score of 1.0 = there are good books on how to use the software; .9 = there are technical reports and/or articles on how to use the software; .8 = there is adequate user documentation, perhaps provided by a website and/or online help; 0.5 although there is documentation, the software has a steep learning curve and is difficult to learn without training or study; 0.1 = there is little or no user documentation–users guess.

Source Documentation Source code comments and technical documentation on the implementation

are essential for a piece of open source software to be maintained and extended.

Rubric: A score of 1.0 = there are books or book-level documents on the implementation, which is heavily commented; .9 = there is somewhat complete documentation beyond comments, such as requirements and/or design documents; .8 = there is typical code comments, including header comments; 0.1 = there is little or no source documentation.

License Different open source licenses vary as to how open they really are. There is some minimal definition of "open source" that a license must comply with, but beyond that, public domain is more open than BSD which is more open than the GNU license.

Rubric: A score of 1.0 = public domain; .9 = more liberal than GNU; BSD-style or AT&T-style licenses; .8 = GNU style licenses.

Permanence and accessibility of repository "Openness" spans time. Will a piece of open source software still be available when its inventor dies and stops paying an ISP? Is it at the mercy of some university administrators? Real open source software is widely mirrored and unlikely to get lost.

Rubric: A score of 1.0 = the source code is easily obtained from many third-party mirror sites; .9 = neutral, third-party open source repository; .8 = software is hosted at 1-2 major sites, mainly the developers' institution; .7 = software is hosted at a canonical site from a small institution of unknown permanence; .4 = software is found mainly by googling for it; it moves frequently or is taken down by authorities.

5 Case Studies

The case studies presented in this paper represent examples from a range types and sizes of open source software projects. If your favorite open source program is not present, by all means please rate it according to the rubric and see how it measures up. If you send us your results, we reserve the right to use them in a future study.

5.1 GALib

The popular genetic algorithms library GAlib from MIT (lancet.mit.edu/ga/) makes an excellent subject of study. It is 21,000 lines of C++, compiles on UNIX, Windows, and MacOS. It is commented moderately well. It is distributed only through a university website. How does it fare:

Language portability 1.0 – C++ is available everywhere.
Library portability 0.9 – GAlib does not depend on other libraries, except for the optional graphics.
Contributors 0.5 – GAlib is Cathedral style, with user bug reporting
Users 0.7 – GAlib is widely used within a niche (approximately 100).
Build Documentation 1.0 – GAlib is properly documented and easy to build
User Documentation 1.0 – GAlib has thorough user documentation.
Source Documentation 0.8 – GAlib has header comments and occasional explanations of interesting code.
License 1.0 – the GAlib license is less restrictive than the GNU license
Repository 0.4 – GAlib remains publicly available only by the grace of a private institution.

OFR = .77

Earlier versions of GAlib had build issues; relatively trivial oversights in distribution packaging are showstoppers for third parties.

5.2 Unicon

The Unicon programming language from unicon.org is perhaps 50,000 lines of C. It compiles on UNIX, Windows, and MacOS. It is commented well. It is distributed through Source Forge, although the test versions of source and Windows binary distributions are routinely available from a university website.

Language portability 1.0 – C is available everywhere.
Library portability 0.7 – Unicon requires libraries that it provides itself (gdbm, xpm, libtp) and links to third party libraries if available.
Contributors 0.8 – Unicon has about 12 developers with commit authority, with user bug reporting.
Users 0.7 – Unicon is used within a niche (around 100 users).
Build Documentation 0.8 – Unicon requires a configuration step, and needs its bin/ directory to be added to the PATH.
User Documentation 1.0 – Unicon has thorough user documentation.
Source Documentation 1.0 – the source is heavily commented, mostly thanks to Ralph Griswold and his students who wrote Icon. There are books and technical reports available.
License .8 – Unicon uses the GNU license.

Repository 0.9 – Unicon uses Source Forge, which is near-permanent.

OFR = .85

5.3 Alice

The Alice 3D animation environment and semi-visual programming language from alice.org is mostly written in Java, a very portable language, with a little bit in Python, also a quite portable language. Judging from the 2.0/2.2/3.0beta implementations' performance on Macintosh and Linux, these portable languages are not as portable as is sometimes claimed.

Language portability 0.9 – Java is available everywhere, but version dependencies aside, the software seems stable mainly on Windows.
Library portability 0.9 – Alice depends on fonts that do not view very nicely on UNIX/Linux.
Contributors 0.4 – Alice uses closed source ivory tower development, with user bug reporting.
Users 0.8 – Alice is popular with thousands of users, mainly in the field of education.
Build Documentation 0.1 – Alice does not include build documentation.
User Documentation 1.0 – Alice has marvelous user documentation
Source Documentation 0.1 – the source is barely commented.
License 0.9 – Alice uses a more liberal than GNU license.
Repository 0.8 – The source repository is based at CMU and we are almost at its mercy. An obsolete version (2.0) is hosted on code.google.com.

OFR = .49

5.4 Linux kernel

The Linux kernel is very large and written in a mixture of C and assembler.

Language portability 0.8 – C is available everywhere, but assembler is not already written for every CPU.
Library portability 1.0 – an OS is complete by definition.
Contributors 1.0 – Linux has thousands of contributors.
Users 1.0 – Linux has millions of users.
Build Documentation 1.0 – Linux has fabulous build documentation.
User Documentation 1.0 – Linux has thorough user documentation.
Source Documentation 0.95 – the source is heavily commented, but who wants to read all that low-level code?
License .8 – Linux uses the GNU license.
Repository 1.0 – Linux is massively mirrored.

OFR = .95

5.5 Open Office

OpenOffice.org is an Office Productivity Suite written in C++ with hundreds of thousands of lines of code. It runs under many operating systems, including Microsoft Windows, Linux, Solaris, BSD, OpenVMS, OS/2 and IRIX. It is commented – though quality varies across the large project. It is distributed mainly through the OpenOffice.org web site.

Language portability 1.0 – C++ is available everywhere.
Library portability 1.0 – The Open Office project provides libraries as part of its download.
Contributors 1.0 – Open Office has many developers, is open to contributions, and includes user bug reporting.
Users 1.0 – Open Office is used widely by millions of people.
Build Documentation 1.0 – Open Office has extensive build documentation.
User Documentation 1.0 – Open Office has extensive user documentation.
Source Documentation 0.9 – the source is commented, though quality varies across the large project.
License 0.8 – Effective with OpenOffice.org 3.0 Beta the Open Office project uses the GNU Lesser General Public License v3 (LGPL).
Repository 0.7 – Open Office uses its own Web site, although it appears that it is in fact widely mirrored.

OFR = .93

5.6 LaTex

LaTeX is a document markup language and document preparation system especially well suited for the preparation of scientific papers and books. It and the underlying TeX system have survived complete reimplementation and extensive revision multiple times.

Language portability 1.0 – LaTex can be ported to just about any operating system.
Library portability 1.0 – Libraries are available for just about any operating system.
Contributors 0.8 – Different distributions are available. However it seems that LaTeX follows a cathedral style software model.
Users 1.0 – LaTeX is used widely by millions of people
Build Documentation 1.0 – LaTex has extensive build documentation.
User Documentation 1.0 LaTex has extensive user

documentation.
Source Documentation 1.0 – the source is commented well.
License 0.5 – is distributed under a free software license, the LaTeX Project Public License (LPPL). The LPPL is not compatible with the GNU General Public License and is therefore somewhat controversial. For example, the Debian Linux community considered excluding LaTeX from its core distribution.
Repository 1.0 – LaTeX code and development is widely mirrored.

OFR = .90

5.7 OpenSolaris

Solaris is an excellent example of a widely used commercial software whose owners found it expedient to open the source.

Language portability 0.7 – OpenSolaris is mostly C and above but includes assembler for a small range of CPUs, including x86 and sparc.
Library portability 1.0 – As an operating system, it depends on no third party libraries.
Contributors 0.9 – OpenSolaris is organized into communities around core technologies, which encourages participation.
Users 1.0 – it is murky trying to distinguish OpenSolaris users from Sun's commercial Solaris users, but it counts as a large audience.
Build Documentation 0.7 – building an OS is complicated; more so when its instructions start by telling you your compiler probably will not work.
User Documentation 1.0 – Numerous titles including an adaptation of Sobell's popular UNIX book suggest that OpenSolaris is well-documented.
Source Documentation 1.0 – A two-volume internals book set on OpenSolaris is available from Prentice-Hall.
License 0.9 – OpenSolaris uses a license based on the Mozilla license. It is somewhat less restrictive than the GPL.
Repository 1.0 – Repositories on three sites use three different protocols.

OFR = .90

5.8 SecondLife

Second Life is a popular multi-user client-server virtual world building system developed by Linden Labs. Like OpenSolaris, it began as a proprietary program and was migrated to open source. Note that only the client has been open sourced; the server was widely announced as being open sourced, and then Linden backed off on their plans. The server constitutes the "keys to the kingdom", since Linden derives most of their revenue from their servers and the virtual property that they host.

Language portability 1.0 – SecondLife is written in C++, quite portable.
Library portability 0.2 – SecondLife viewer source distribution includes a binary .dll file, a shared library of unknown contents. Worse, it is not usable without a server whose source code has not been released.
Contributors 0.5 – It is not clear how many persons outside Linden Labs have contributed to SecondLife.
Users 1.0 – Second Life has millions of users.
Build Documentation 0.7 – The SecondLife viewer build is non-trivial.
User Documentation 1.0 – There are many books on Second Life.
Source Documentation 0.7 – There are comments in the code, but not a lot.
License 0.8 – GPL.
Repository 0.8 – The software is hosted at Linden Labs.

OFR = .68

5.9 Freespire

Freespire is a Linux distribution, which came about after the closing of the ill-fated Linspire and Lindows projects. Freespire seeks to make Linux easy on the PC, mainly by including proprietary codecs, drivers and applications.

Language portability 0.8 – Freespire uses a Debian kernel; it is as portable as Linux.
Library portability 0.8 – Freespire uses proprietary libraries in part.
Contributors 0.7 – Freespire has a somewhat limited contributor base.
Users 0.7 – Freespire has a somewhat limited user base.
Build Documentation 0.7 – Freespire's build documentation is not as extensive as for other distributions.
User Documentation 0.8 – Freespire's user documentation is not as readily available as for other distributions.
Source Documentation 0.5 – there are proprietary segments of the projects.
License 0.8 – Open, except for proprietary parts.
Repository 0.5 – The distribution is not widely mirrored.

OFR = .69

5.10 MediaPortal

MediaPortal (from www.team-mediaportal.com) is an example of an important category of open-source software: platform-specific. MediaPortal only runs on Windows, even though it is a "fork" of a multi-platform open source project, XBMC.

Language portability 0.86 – Written in a mixture of C# and C++, the former being a platform-specific niche language, albeit a nice one. The number would be lower, but there is a free C# Express compiler available so long as you are a Windows developer. 86% is taken from the current Windows market share.
Library portability 0.7 – MediaPortal uses an enormous number of libraries, including Windows-only libraries and Net 2.0. It complains of missing hotfixes on mainstream Windows versions.
Contributors 0.8 – On the Source Forge site, 69 accounts are associated with the project, however, many are non-developers.
Users 0.7 – The Source Forge site lists an approximate subscriber count of 52 for the MediaPortal mailing list.
Build Documentation 0.3 – There are build instructions available. The software is very large and complex to download or build. No "source distribution" is available other than the SVN repository. The SVN repository includes many binary .exe and .dll files, making it unclear whether it is complete in source form.
User Documentation 0.9 – There are articles available on using MediaPortal.
Source Documentation 0.8 – MediaPortal includes normal code comments.
License 0.8 – MediaPortal is under the GPL.
Repository 0.9 – Source Forge.

OFR = .72

6 Openness Categories

According to the OFR numbers given in Section 5, the highest openness of any tool measured is the Linux kernel with a score 0.95 and the lowest project is the Alice project with a score of 0.49. We propose to introduce openness classes as follows: OFR greater than 0.9 is class I, OFR greater than 0.75 is class II, and OFR less than 0.75 is class III. According to this designation, Linux and OpenOffice are class I, GAlib and Unicon are class II, and Alice, SecondLife, Freespire and MediaPortal are class III open source software.

Figure 1 illustrates our categories further. Assume that a category C_i is considered to have a "passing" grade if its score is above $\frac{1}{2}$, else it is considered "failing." In the figure we show the geometric progression as the number of failing scores increases. More than two failing scores will put a project into class II. More than four failing scores result in class III evaluation.

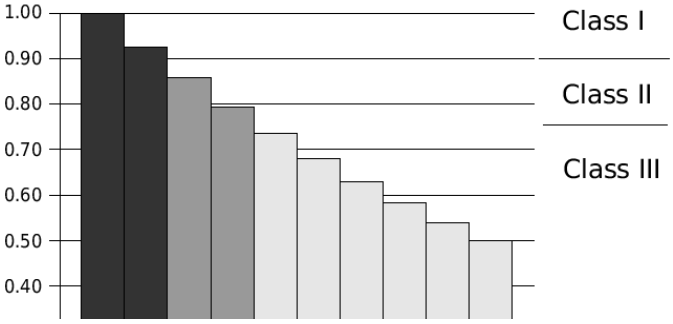

Figure 1. Class Digression for $C_i = \frac{1}{2}$

The geometric nature of our scoring system makes it possible to readily assign an OFR to a combined project. Say project $\mathcal{A}$ has $\text{OFR}_{\mathcal{A}} = a$ and project $\mathcal{B}$ has $\text{OFR}_{\mathcal{B}} = b$ then the combined score for the combined project $\mathcal{A}, \mathcal{B}$, $\text{OFR}_{\mathcal{A},\mathcal{B}}$ can be calculated as

$$\text{OFR}_{\mathcal{A},\mathcal{B}} = \sqrt{a \cdot b}.$$

This assumes that both projects have roughly equal weight. If the projects are weighted by weights α and β then we have

$$\text{OFR}_{\mathcal{A},\mathcal{B}} = (a^{\alpha} \cdot b^{\beta})^{\frac{1}{\alpha+\beta}}.$$

7 Conclusions

Although the proposed OFR classification system is relatively simplistic and straightforward it does seem to separate open source projects out into the sheep and goats. During our research work we were somewhat surprised that many of the case studies (some of them not included, eg. Firefox, Gimp, Chromium) received high marks under the OFR system. The fact that many of the presented examples fall into the "closed" open source Category III is skewed by our looking specifically for open source posers. The majority of the open source community largely adheres to the open source rules and practices. The Category III projects generally stand out as not very open projects, although some are worse than others. It will be valuable to identify more such open source posers. The authors wish to reiterate their respect for both closed source and cathedral-style development. We hope, however, that the OFR measure will help encourage some open source projects to be more open.

Further work could focus on better quantifying measures for each of the nine categories. Certainly new categories could be introduced. One could also reflect on weighing categories, perhaps depending on the type of open source project, given that the project "Linux kernel" is quite different from the project "LaTex" or the project "Unicon". We finally note that Freespire is really the concatenation of two projects, namely a relatively open Debian project and a project that has many proprietary parts. Thus the rating value of 0.72 could have been derived using the concatenation formula given at the end of Section 6.

References

[1] Martin Aigner and Günther M. Ziegler. *Proofs from The Book*. Springer Verlag, 3rd edition, 2004.

[2] R. Glass. A look at the economics of open source. *Communications of the ACM*, 47:25–27, 2004.

[3] Daniel E. Markle. Community and "Fake" Open Source Projects. `www.ashtech.net`, 2005.

[4] Eric S. Raymond. *The Cathedral and the Bazaar*. O Reilly, 1999.

[5] Todd R. Weiss. Open-source group to create online forum for corporate users. Computerworld, 2007.

[6] Nathan Willis. When open source projects close the process, something's wrong. `www.linux.com/archive/feature/120635`, 2007.