UPGRADE
The European Journal for the Informatics Professional
http://www.upgrade-cepis.org
Vol. IV, No. 4, August 2003

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

Publisher
UPGRADE is published on behalf of CEPIS (Council of European Professional Informatics Societies, <http://www.cepis.org/>) by NOVÁTICA <http://www.ati.es/novatica/>, journal of the Spanish CEPIS society ATI (Asociación de Técnicos de Informática <http://www.ati.es/>).
UPGRADE is also published in Spanish (full issue printed, some articles online) by NOVÁTICA, and in Italian (abstracts and some articles online) by the Italian CEPIS society ALSI <http://www.alsi.it> and the Italian IT portal Tecnoteca <http://www.tecnoteca.it/>.
UPGRADE was created in October 2000 by CEPIS and was first published by NOVÁTICA and INFORMATIK/INFORMATIQUE, bimonthly journal of SVI/FSI (Swiss Federation of Professional Informatics Societies, <http://www.svifsi.ch/>).

E-mail addresses for editorial correspondence: <nicolet@acm.org> and <rfcalvo@ati.es>

E-mail address for advertising correspondence: <novatica@ati.es>

Upgrade Newslist available at <http://www.upgrade-cepis.org/pages/editinfo.html#newslist>

ISSN 1684-5285

Next issue (Oct. 2003):
"e-Learning – Borderless Education"

Software Engineering – State of an Art

Guest Editor: **Luis Fernández-Sanz**

Joint issue with NOVÁTICA

Free Software Engineering: A Field to Explore

Jesús M. González-Barahona and Gregorio Robles

The challenge of free software is not that of a new competitor who, under the same rules, produces software faster, cheaper and of a better quality. Free software differs from 'traditional' software in more fundamental aspects, starting with philosophical reasons and motivations, continuing with new economic and market rules and ending up with a different way of producing software. Software Engineering cannot ignore this phenomenon, and the last five years or so has seen ever more research into all these issues. This article takes a look at the most significant studies in this field and the results they are producing, with a view to providing the reader with a vision of the state of the art and the future prospects of what we have come to call free Software Engineering.

Keywords: Software Engineering, free Software Engineering, free software.

1 Introduction

Although free software[1] has been developed for several decades, only for the last few years have we begun to pay attention to its development models and processes from the point of view of Software Engineering. Just as there is no single development model for proprietary software, neither is there only one in the free software world [11], but, that said, some interesting features are shared by a large number of the projects we have looked at, features which may be at the root of free software.

In 1997 Eric S. Raymond published his first, widely read article "The cathedral and the bazaar" [18], in which he describes some characteristics of free software development models, laying great stress on what differentiates these models from proprietary development models. Since then that article has become one of the best known (and most criticised) in the free software world, and to a certain extent, was the starting pistol for the development of free Software Engineering.

2 The Cathedral and the Bazaar

Raymond makes an analogy between the way mediaeval cathedrals were built and the classic way of producing software. He argues that in both cases there is a clear distribution of tasks and roles, with the designer on top of everything, controlling the process from beginning to end. Planning is strictly controlled in both cases, giving rise to clearly defined processes in which, ideally, everyone taking part in the activity has a very limited and specific role to play.

Included in what Raymond views as the cathedral building model are not only the heavyweight processes of the software industry (the classic cascade model, the different aspects of the Rational Unified Process, etc.), but also some free software projects, as is the case of GNU, <http://www.gnu.org/>, and NetBSD, <http://www.NetBSD.org/>. According to Raymond, these projects are highly centralized, since only a few people are responsible for the design and implementation of the software. The tasks these people perform and the roles they play are perfectly defined, and anyone wanting to join the development team would need to be assigned a task and a role according to the needs of the project. Another feature is that releases of this type of programme tend to be spaced out over time, following a fairly strict schedule. This leads to having few releases of the software with lengthy intervals between each

Jesús M. González Barahona is a professor at the Universidad Rey Juan Carlos, Madrid, Spain. He researches in the field of distributed systems and large scale peer to peer computing. He is also interested in free / open source Software Engineering. He began working on the promotion of free software in 1991. He is currently collaborating on several free software projects (including Debian), and he collaborates with associations such as Hispalinux and EuroLinux, writes in several media on free software related matters, and advises companies on their strategies related to these issues. He is a member of ATI, coordinator of the Free Software section of Novática, and has also been a guest editor of several monographs in Novática and Upgrade.
<jgb@gsyc.escet.urjc.es>

Gregorio Robles is a professor at the Universidad Rey Juan Carlos, Madrid, Spain. His research work is centred on the study of free software development from an engineering point of view and especially with regard to quantitative issues. He has developed or collaborated on the design of programmes to automate the analysis of free software and the tools used to produce them. He was also involved in the FLOSS study of free software financed by the European Commission IST programme.
<grex@gsyc.escet.urjc.es>

1. **Note from the Editor of Upgrade:** Our editorial policy is to continue to use, in English, the term 'free software', though we are aware that the term 'open source software', or simply 'open source', appears to be winning the battle; 'libre software' is also gaining popularity because it avoids the ambiguity of the English word 'free'.

one, and means that the programmes have clearly defined stages.

The opposite to the cathedral model is the bazaar. According to Raymond, some free software programmes, especially the Linux kernel, have been developed along similar lines to an oriental bazaar. In a bazaar there is no governing authority controlling the processes being developed or carefully planning each move. And the roles of those taking part can change constantly (sellers can become customers) without any instructions from outside.

But perhaps the most novel thing about Raymond's book [18] is how he describes the process that has turned Linux into a success story in the free software world; it is a series of 'good methods' to take full advantage of the possibilities afforded by the availability of source code and interactivity through the use of telematic systems and tools.

A free software project tends to be born as a result of a purely personal action; it starts with a developer who finds him or herself unable to solve a problem fully. The developer must have the knowledge required to at least begin to solve it. Once he or she has managed to get something usable, simple, with some functionality, if possible, well designed or written, the best thing they can do is to share that solution with the free software community. This is what is known in the free software world as releasing early and it has the effect of attracting the attention of other people (generally developers) who have the same problem and who may be interested in the solution.

One of the basic principles of this development model is to consider users as co-developers. They need to pampered, not only because they can provide publicity by word of mouth, but also because they will carry out one of the most expensive tasks that there is in software production: the testing. Unlike co-development, which is not readily scalable, debugging and testing are by nature highly parallelable. It's the user who takes the software and tests it on his or her machine under specific conditions (an architecture, some tools, etc.), a task which, multiplied by a large number of architectures and environments, would require a major effort from the development team.

If users are treated as developers then there is the possibility that one of them will find an error, correct it, and send a patch to the developer of the project so that the problem will be solved in the next version. It may also happen that, for example, the problem is eventually understood and corrected by a different person from the one who original pinpointed the error. Whatever the case, all of these scenarios are clearly very advantageous to the development of the software, and a dynamic of this nature is highly beneficial to software development.

This situation becomes even more effective with frequent releases, since the motivation to find, notify and correct errors is high as problems are seen to be addressed immediately. There are also spin-off advantages, such as the fact that as integration is performed frequently – ideally at least once a day – a final phase of integration of the modules making up the programme is rendered unnecessary. This principle is known as releasing often and allows for great modularity [17] while maximizing the publicity effect of releasing a new version of the software. As can be seen in [5], release management in large scale projects tends to follow a well-defined and somewhat complex process.

To prevent frequent releasing from putting off users who value stability over speed of software evolution, some free software projects have several branches of development running in parallel. The best known case is the Linux kernel, which has a stable branch – aimed at those people who value its reliability – and another, unstable branch – targeting developers – with the latest innovations and features.

3 Leadership and Decision Taking in the Bazaar

Raymond assumes that every free software project must have a 'benevolent dictator', a kind of leader – who is usually the project's founder –, responsible for guiding the project and who always has the last word at decision taking time. The skills this person needs to have are mainly the ability to motivate and coordinate a project, understand the users and co-developers, reach consensuses and integrate anything that might contribute something to the project. Readers may notice that we haven't mentioned technical competence as one of the most important requirements, though that wouldn't come amiss either.

With the growth in size of and number of developers involved in some free software projects, new ways of organizing the decision taking process have begun to emerge. Linux, for example, has a hierarchical structure based on the delegation of responsibilities by Linus Torvalds, the 'benevolent dictator' (see <http://www.linux.org/>.) This leads to a situation where there are parts of Linux with their own 'benevolent dictators', although their 'power' is limited by the fact that Linus Torvalds will always have the last word. This case is a clear example of how the high degree of modularity that exists in a free software project has given rise to a specific organizational and decision taking process [17].

The Apache Foundation, <http://www.apache.org/>, on the other hand is more of a 'meritocracy', since it has a management committee made up of people who have made significant contributions to the project. It is not in fact a meritocracy in the strictest sense of the word, one in which those who contribute the most have most say in what goes on, since the management committee is democratically elected every so often by members of the Apache Foundation (who are responsible for the management of several free software projects, including Apache, Jakarta, etc.). To become a member of the Apache Foundation you need to have made a continuous and major contribution to one or more projects of the foundation. This system is also used in other large scale projects, such as FreeBSD, <http://www.freebsd.org/>, or GNOME, <http://www.gnome.org/>.

Another interesting instance of formal organization is the GCC Steering Commitee, <http://gcc.gnu.org/steering.html>. It was created in 1998 to prevent anyone from taking control of the GCC project (GNU Compiler Collection, GNU's compilation system) and backed by the FSF (Free Software Foundation, <http://www.fsf.org>), a promoter of the GNU project, a few months later. In a sense it is a committee which follows in the tradition of a similar one governing the egcs project (which

for a time ran parallel to the GCC project, but later came to join it). Its basic mission is to ensure that the GCC project respects its own 'Mission Statement'. The committee members are elected on an individual basis by the project itself, so that to a greater or lesser extent they represent the various communities collaborating in the development of GCC (language support developers, kernel developers, groups interested in embedded programming, etc.).

A person's leadership of a free software project does not have to be carved in stone. There are two basic reasons why a project leader should stand down. The first reason is through lack of interest, time or motivation to carry on. In this case he or she should pass the baton to another developer to take on the role of project leader. Recent studies [7] show that the leadership of projects tends to change hands frequently, with several 'generations' of developers leading projects over a period of time. The second reason is more problematical: code forking. Free software licences allow anyone to take the code, modify it and redistribute it without the approval of the project leader. This does not tend to happen as a rule, except in cases when it is done deliberately to outflank the project leader (and prevent his or her possible veto of a contribution). This is a clear case of a kind of 'coup d'état', however legal and legitimate such an action might be. For this reason one of the main reasons why a project leader tries to keep his co-developers happy is to minimize the risk of code forking.

4 Processes in Free Software

Although free software is not necessarily associated with any specific software development process, there is a broad *consensus* on the most common processes used. That is not to say that there are not free software projects created using classic processes such as the cascade model. But generally speaking, the development model of free software projects tends to be more informal, largely due to the fact that the development team performs the tasks voluntarily and for no financial gain, at least none of a direct nature.

The way requirements are captured in the free software world depends both on the 'age' and the size of the project. In the early stages, the project founder and user tend to be one and the same person. Later, if the project grows, requirements capture tends to take place via e-mailing lists and there is usually a clear distinction between the development team – or at least the most active developers – and the users. For major projects, with a great many users and developers, requirements are captured via the same tool used for error management. In this case, we talk about activities rather than errors, although the mechanism used to manage them is the same as is used for error correction (they are classified by importance, dependence, etc. and can be used to monitor whether errors have been corrected or not). The use of this planning tool is a relatively recent innovation, and is a clear indication that in the free software world there has been an evolution from a total lack of any management system to a centralized system of engineering management, albeit a fairly limited one. However, the typical requirements capture document of a cascade model is not normally seen

As for a system's overall design, normally this is only exhaustively documented for major projects. For smaller projects, the main developer or developers will probably be the only ones in possession of the overall design – often only in their heads – or it will come together in later versions of the software. The lack of any overall design does not only mean limitations in terms of the possible reuse of modules, but it is also a great handicap when it comes to inviting in new developers, since they will have a slow and costly learning curve ahead of them. And it isn't so easy to find detailed design either, which means that a great many opportunities to reuse code are lost.

It is in the implementation phase where free software developers concentrate most of their efforts, mainly because, in the eyes of developers, this is where most fun is to be had. In this phase the prevalent programming paradigm tends to be the classic one of trial and error until the desired results are achieved from the subjective point of view of the programmer. In the past, unit tests have rarely been included with the code, even though this would have been an aid to other developers needing to modify existing code or include new code. In the case of some large scale projects, Mozilla for example, there are machines dedicated exclusively to downloading the repositories containing the most up to date code in order to compile it for different architectures [19]. The errors detected are posted on a developers' mailing list.

However, automatic testing is not very well established. Normally it will be the users themselves who perform the testing, within a great variety of uses, architectures and combinations. This has the advantage of parallelizing at a minimum cost to the development team. The downside of this model is the problem of how to organize things so that there is user feedback, and that this feedback is as efficient as possible. With regard to maintenance of software in the free software world – by this we mean the maintenance of earlier versions– this task may or may not exist depending on the project. In projects requiring great stability, such as operating system kernels, etc., earlier versions of the project are maintained, since a change to a newer version can be traumatic. But, as a rule, in most free software projects, if there is an earlier version and an error is found in it, the developers tend not to try and correct it, but instead advise users to use the most up to date version in the hope that it will have disappeared as a result of the software's evolution.

5 Criticism of the Cathedral and the Bazaar

Raymond's book [18] suffers from a lack of systematicity and rigour, to be expected from its somewhat essayistic and admittedly not very scientific nature. The most frequent criticisms refer to the fact that it is basically describing one isolated experience – Linux – and tries to extrapolate the conclusions reached to all free software projects, when, as can be seen in [14], the existence of such an extensive community as the Linux kernel has, is the exception rather than the rule.

More critical still are those who think Linux is an example of a development which follows the cathedral model of development. They argue that there is obviously an intellectual leader,

or at least someone at the top in power, and a hierarchical system through delegation of responsibilities right down to the worker-programmers. There is also a division of tasks, albeit implicitly. In [2] the criticism goes further, maintaining – not without a certain degree of acrimony and arrogance in the reasoning – that the metaphor of the bazaar is self-contradictory.

Another of the most criticized points in Raymond's book [18] is his claim that Brooks's law, which states that "*Adding manpower to a late software project makes it later*" [4] is not valid in the free software world. In [12] we can read that the truth of the matter is that we are talking about a different environment, and what at first sight appears to be an incongruity with Brooks's law is, after a more exhaustive study, merely an illusion.

6 Quantitative Studies

Free software enables developers to make full use of the quantitative analysis of code and all the other parameters involved in its production, thanks to their accessibility. This may benefit traditional Software Engineering areas – such as empirical Software Engineering – due to the existence of a massive amount of information that can be accessed without the need for any major interference with free software development. The authors are convinced that this vision may be of enormous assistance in the analysis and understanding of the phenomena involved in the creation of free software (and of software in general) and, among other possibilities, may even lead to us having predictive models of software with real time feedback.

The idea behind this is very simple: as we have the chance to study the evolution of a huge number of software projects, let's do it. Especially since not only is the current state of a project public, but so is all its past development, which means that all this information can be extracted, analysed and packaged to be used as a knowledge base allowing us to assess a project's 'health', facilitate decision taking and predict current and future complications.

The first quantitative analysis of some importance in the free software world dates back to 1998, although it was only published in early 2000 [10]. Its purpose was to gain empirical knowledge of the participation of developers in the free software world. To this end they made a statistical study of the authorship attribution which writers tend to include in their source code header files. It was shown that developers' participation followed Pareto's law: 80% of the code corresponds to the 20% most active developers, while the remaining 80% of developers contribute 20% of the total. Many later studies have confirmed this and extended the validity of the result to other forms of participation apart from source code contribution (messages to mailing lists, error notification, etc.). The tool used to carry out this study was published by the authors under a free licence, thereby providing the possibility of reproducing its results or performing new studies.

In a later study, Koch [13] took things a step further and also analysed the interactions carried out in a free software project. The source of information in this case was the mailing list and the version repository of the GNOME project. But the most interesting aspect of Koch's study is his economic analysis. Koch focuses on checking the validity of classic cost prediction (function points, COCOMO – COnstructive COst Model – model, etc.) and shows the problems involved in its application, although he does admit that the results obtained –used with due caution – are at least partially reliable. He concludes that free software requires its own study methods and models, since the ones we have now are not suitable. However, it would seem obvious that the chance to obtain a great deal of the data related to the development of free software publicly should allow us to be optimistic that in a not to distant future such methods and models will be available. Koch's study can be considered as the first complete quantitative analysis, even though it may lack a clear methodology and, above all, ad hoc tools enabling results to be checked and other projects to be studied.

In the year 2000 Mockus et al. presented the first study of free software projects to include the complete description of the development process and organizational structures, including both qualitative and quantitative evidence [16]. To do this they used the log of changes to the software and the error reports to quantify such aspects as the participation of developers, the size of the core, the authorship of the code, productivity, defect density and error resolution intervals. In a sense, this study is still a classic Software Engineering study, except for the fact that the data collection was carried out entirely by means of the semi-automatic inspection of data that projects offer publicly on the Internet. As in [13], this article provides no tool or automatic process to allow the method to be reused in the future by other research teams.

[23] and [24] deal with the quantitative analysis of the lines of code and programming languages used in the Red Hat distribution, <http://www.redhat.com/>. Barahona et al. have followed in their footsteps in a series of articles on the Debian distribution, <http://www.debian.org/> (see for example [6] and [8]). All of them provide a kind of 'x-ray' of these two popular GNU/Linux distributions using data supplied by a tool which counts the number of physical lines (the lines of code which are neither blank nor commentaries) in a programme. Apart from the spectacular result in terms of total lines of code (the latest stable version to date, Debian 2.2 – known as Woody – has over one hundred million lines of code), you can also see the distribution of the number of lines in each programming language. The possibility of studying the evolution of the different versions of Debian over time throws up some interesting results [8], among which is the fact that the average size of packets has remained practically constant over the last five years, which means that the natural tendency of packets to grow has been neutralized by the inclusion of smaller packets. It is also possible to see how the importance of the programming language C, while still predominant, is decreasing over time, while script languages such as Python, PHP and Perl, and Java are showing an explosive growth. 'Classic' compiled languages (Pascal, Lisp, Ada, Modula...) are clearly in decline. Finally, these articles include a section which shows the results obtained if the COCOMO model – a classic effort estimation model dating back to the early eighties [3] and used in proprietary software

projects – is applied to perform an estimation of effort, project duration and cost.

Although they are forerunners, most of the studies presented in this section are more or less limited to the projects they analyse. The methodology used has been designed to suit the project being studied, is partially manual and only on rare occasions can the automated part be used in any general way in other free software projects. This means that a very great effort is required to study any new project, since the method used needs to be adapted and all the manual actions performed need to be repeated.

For this reason the latest efforts ([20] or [9]) have concentrated on creating an analysis infrastructure integrating several tools in such a way that the process is as automated as possible. There are two very obvious reasons for doing this. The first is that once a lot of time and effort has been invested in creating a tool to analyse a project – putting great stress on its genericity –, using it for other free software projects involves the minimum of effort. And secondly, analysis by means of a series of tools which analyse programmes from different points of view – sometimes complementary to one another, other times not – allows us to get a broader perspective of the project. In [15] these initiatives can be studied in greater depth.

7 Future Work and Conclusions

After this brief but intense history of Software Engineering applied to free software, we can say that free Software Engineering is still in its infancy. There are several very important aspects still needing to be carefully studied and examined in great depth in order to come up with a model explaining, if only in part, how to create free software. The issues that need to be addressed in the near future are the following: the classification of free software projects, the creation of a methodology based as much as possible on automated analysis and the use of the knowledge thus acquired to create models enabling us to understand free software development while facilitating decision taking based on acquired experience.

Currently one of the most important weaknesses is the lack of any strict classification of free software enabling projects to be put into different categories. At the moment classification criteria are too rough and ready: projects having widely disparate organizational, technical, and other kinds of features are all lumped together. The argument that Linux, with an extensive developer community and a large number of users, is by nature different and behaves differently from a much more limited project in terms of number of developers and users, cannot be denied. What is clear is that more exhaustive classification would enable the experience acquired in similar projects (that is to say, with similar characteristics) to be reused, would make predictions easier, would enable risk forecasting, etc.

The second important issue that free Software Engineering has to address, closely linked to the point above and to current trends, is the creation of a methodology and tools to support it. A clear and concise methodology will enable us to study all projects using the same yardstick, to verify their current state, know their evolution and, of course, classify them. Tools are essential for tackling this problem, since, once created, they will allow us to analyse thousands of projects with a minimal additional effort. One of the aims of free Software Engineering is that from a small number of parameters indicating where to find information about the project on the net (the address of the software version repository, the place where the mailing list archives are stored, the location of the error management system and a brief survey) an exhaustive analysis can be made of the project. Project managers would only be one button away from a complete analysis, a kind of 'clinical analysis' which would enable them to judge the project's 'health' and have an indication of the aspects that needed improving.

Once we have methods, classification and models, the possibilities afforded by simulation – and, to be more specific, by intelligent agents – could be enormous. Bearing in mind that our starting point is a system of notorious complexity, the creation of dynamic models in which the different agents participating in software production can be modelled is of great interest. Clearly the more we know about the different components, the closer to reality the model will be. Although we are aware of several proposals for simulations for free software, they are a little too simple and incomplete. This is due to a certain extent to the fact that there is still a great dearth of knowledge concerning the interactions which take place in the generation of free software. If project information can be appropriately packaged and processed throughout its entire history, intelligent agents may be crucial to knowing how the project will evolve in the future. Although there are a number of proposals about how this problem needs to be approached, currently one of the most advanced proposals can be found at [1].

To sum up, in this article we have seen how free Software Engineering is a young and as yet largely unexplored field. Its first steps were made in essayistic writings in which a more efficient development model was proposed – not without a certain lack of scientific rigour –, but gradual progress has been made in the systematic study of free software from an engineering viewpoint. Currently, after several years of reports and complete quantitative and qualitative analyses of free software projects, an enormous effort is being made to obtain an overall infrastructure enabling projects to be classified, analysed and modelled in a limited space of time and an at least partially automated way. When the analysis of free software projects ceases to be as costly in terms of time and effort as it is today, an exciting new phase of free Software Engineering is likely to begin, in which another type of techniques will take centre stage, techniques whose main purpose will be to predict the evolution of software and anticipate possible complications.

Translation by Steve Turpin

References

[1] Antoniades Ioannis, Samoladas Ioannis, Stamelos Ioannis, and G. L. Bleris. Dynamical simulation models of the Open Source Development process, 2003, pending publication in Free/Open Source Software Development, published by Stefan Koch, Idea Inc, Vienna.

[2] Nikolai Bezroukov. A Second Look at the Cathedral and the Bazaar, December 1998. <http://www.firstmonday.dk/issues/issue4_12/bezroukov/index.html>.

[3] Barry W. Boehm, 1981. Software Engineering Economics, Prentice Hall.

[4] Frederick P. Brooks Jr., 1975. The Mythical Man-Month: Essays on Software Engineering, Addison-Wesley.

[5] Justin R. Ehrenkrantz. Release Management Within Open Source Projects, May 2003. <http://opensource.ucc.ie/icse2003/3rd-WS-on-OSS-Engineering.pdf>.

[6] Jesús M. González Barahona, Miguel A. Ortuño Pérez, Pedro de las Heras Quirós, José Centeno González, and Vicente Matellán Olivera. Counting potatoes. The size of Debian 2.2, Upgrade, vol. 2, issue 6, December 2001. <http://upgrade-cepis.org/issues/2001/6/up2-6Gonzalez.pdf>. Also available at <http://people.debian.org/~jgb/debian-counting/>.

[7] Jesús M. González Barahona and Gregorio Robles. Unmounting the code god assumption, Mayo 2003, Proceedings of the Fourth International Conference on eXtreme Programming and Agile Processes in Software Engineering (Genoa, Italia). <http://libresoft.dat.escet.urjc.es/html/downloads/xp2003-barahona-robles.pdf>.

[8] Jesús M. González Barahona, Gregorio Robles, Miguel A. Ortuño Pérez, Luis Rodero Merino, José Centeno González, Vicente Matellán Olivera, Eva M. Castro Barbero, and Pedro de las Heras Quirós. Anatomy of two GNU/Linux distributions, 2003", pending publication in "Free/Open Source Software Development, published by Stefan Koch, Idea Inc, Vienna.

[9] Daniel Germán and Audris Mockus. Automating the Measurement of Open Source Projects, May 2003. <http://opensource.ucc.ie/icse2003/3rd-WS-on-OSS-Engineering.pdf >.

[10] Rishab Aiyer Ghosh and Vipul Ved Prakash, The Orbiten Free Software Survey, May 2000. <http://www.firstmonday.dk/issues/issue5_7/ghosh/index.html>.

[11] Kieran Healy and Alan Schussman, The Ecology of Open Source Software Development, January 2003. <http://opensource.mit.edu/papers/healyschussman.pdf>.

[12] Paul Jones. Brooks' Law and open source: The more the merrier?, May 2000, <http://www-106.ibm.com/developerworks/opensource/library/os-merrier.html?dwzone=opensource>.

[13] Stefan Koch and Georg Schneider. Results from Software Engineering Research into Open Source Development Projects Using Public Data, 2000. <http://wwwai.wu-wien.ac.at/~koch/forschung/sw-eng/wp22.pdf >.

[14] Sandeep Krishnamurthy. Cave or Community? An Empirical Examination of 100 Mature Open Source Projects, May 2002. <http://opensource.mit.edu/papers/krishnamurthy.pdf>.

[15] Jesús M. González Barahona and Gregorio Robles Martínez. Libre Software Engineering. <http://libresoft.dat.escet.urjc.es/>.

[16] Audris Mockus, Roy T. Fielding, and James D. Herbsleb. Two Case Studies of Open Source Software Development: Apache and Mozilla, Junio 2000. <http://www.research.avayalabs.com/techreport/ALR-2002-003-paper.pdf>.

[17] Alessandro Narduzzo and Alessandro Rossi. Modularity in Action: GNU/Linux and Free/Open Source Software Development Model Unleashed, May 2003. <http://opensource.mit.edu/papers/narduzzorossi.pdf >.

[18] Eric S. Raymond. The Cathedral and the Bazaar, Musings on Linux and Open Source by an Accidental Revolutionary, May 1997. <http://catb.org/~esr/writings/cathedral-bazaar/>.

[19] Christian Robottom Reis and Renata Pontin de Mattos Fortes. An Overview of the Software Engineering Process and Tools in the Mozilla Project, February 2002. <http://opensource.mit.edu/papers/reismozilla.pdf>.

[20] Gregorio Robles, Jesús González Barahona, José Centeno González, Vicente Matellán Olivera, and Luis Rodero Merino. Studying the evolution of libre software projects using publicly available data, May 2003, Proceedings of the 3rd Workshop on Open Source Software Engineering at the 25th International Conference on Software Engineering. <http://opensource.ucc.ie/icse2003/3rd-WS-on-OSS-Engineering.pdf >.

[21] Ilkka Tuomi. Internet, Innovation, and Open Source: Actors in the Network, 2001. <http://www.firstmonday.dk/issues/issue6_1/tuomi/>.

[22] Paul Vixie. Software Engineering, 1999. <http://www.oreilly.com/catalog/opensources/book/vixie.html>.

[23] David A. Wheeler. Estimating Linux's Size, July 2000. <http://www.dwheeler.com/sloc>.

[24] David A. Wheeler. More Than a Gigabuck: Estimating GNU/Linux's Size, June 2001. <http://www.dwheeler.com/sloc>.

