A Software Engineering approach to Libre Software

GREGORIO ROBLES

The challenge of libre[1] software is not the one of a new competitor producing, under the same rules, software in a faster and cheaper way, and with higher quality. Libre software differs from "traditional" software in more fundamental aspects, beginning with philosophical reasons and motivations, followed by new economic and market guidelines and finishing with a different form of producing software. Software engineers cannot ignore this and for some years now the investigation of all these aspects has intensified. This article tries to introduce the reader into the most relevant technical aspects of libre software from the perspective of the development process (dubbed the Bazaar model; Raymond 1997). From a more quantitative point of view, several perspectives will be offered: the study of source code, of human resources and production costs. Finally, some of the proposals to continue advancing in the engineering research of libre software are presented.

1. Introduction

Libre software has won, without any doubt, a big importance lately. Although its philosophical principles (and first code) date back to the 1980s,[2] it has not been until some ten years ago that, with the proliferation of Internet connections, it has shown itself as a sound development and distribution method – to the point that it is increasingly becoming a threat for the traditional software industry, the latter being based on a closed development model and a distribution method relying on collecting licensing fees.

The "revolution" in the technical and technological methods in some libre software projects – those whose development form has been named the Bazaar model – has drawn the interest of software engineers. They became increasingly curious, seeing how a completely different approach has spurred the production of high quality, and low cost, software in the course of the last years. Even though the most interesting elements for these researchers have been the technical ones, in the author's opinion, the philosophical aspects related to libre software cannot be left aside: con-

---

1 In an attempt to avoid any confusion regarding the meaning of *free* in *free software*, throughout this article, the term *libre software* is used instead. It was chosen because of its meaning pointing towards *liberation*, and not of merely being costless. The term Open Source is refused for its ignorance about the philosophical foundations of what free software meant in the first place.

2 In fact, libre software already existed before the eighties, historically predating proprietary software. Cf., for example, Campbell-Kelly (2003, S. 54). 'Throughout the 1950s, programs were perceived as objects without intrinsic value, or at best with value that were no market mechanisms to realize. *Users got 'free' software from a computer manufacturer, or they found it through a user group*; more often, however, they wrote it in house.' (emphasis added) It was in 1984, however, that Richard Stallman began pursuing the GNU project as an alternative to the proprietary way of software development, distribution and deployment (Levy 2001, S. 427).

cepts like community and ethics are of great importance. Interpreting and understanding the observations, and answering apparently technical-only questions, requires to take into account this particular philosophy of software development, as we will see later.

There exists an ample confusion that we should clarify at the very beginning of this text: libre software and the Bazaar model are not synonymous. The freedom in software is a legal aspect that becomes reflected in the software license containing the terms and conditions under which the author has published, and allows to distribute, his work. On the other hand, we can locate the Bazaar model in the technical area. It is a way of developing software that promotes opening the development process as much as possible, thus offering the possibility to participate in development and distribution to anyone who desires to.

It is well known that many software projects with a free license follow a Bazaar development model, but this does not mean that all of them do so. Also, without doubt, it is the freedoms in libre software that facilitates carrying out a Bazaar development model. Perhaps, granting these freedoms is even the only way to make the Bazaar possible. In any case, unless it is indicated otherwise, the libre software which we are going to consider in this article will be the one that is produced following a Bazaar model. From a software engineering point of view, this is the one that is of the highest interest.

Clearly, since we are only going to consider Bazaar-style projects, we focus on a subgroup of all libre software projects. It is important to point out that, by doing this, we do not address the great majority of libre software projects which, according to Healy and Schussman (2003), show little activity and are organized in an individual way. In comparison, libre software projects following the Bazaar model are less numerous, but have (in general) a larger number of developers and users as well, what explains its broad diffusion and gives it a bigger impact on the software world.

The structure of this article is as follows: first the characteristics of the Bazaar model will be presented. Next a series of mostly empirical studies that have tried to clarify how the Bazaar works in practice will be discussed. This part has been divided in three sections: source code related technical analysis, analysis of the human resources that take part in the production of libre software and, finally, economic analyses from the engineering point of view. In order to finalize, we try to anticipate the future evolution of libre software research, indicate tools and methods that already are beginning to be used and argue which could be the following steps.

2. Characteristics of the Bazaar

The Bazaar development model received its name from the famous article by Eric S. Raymond (1997). In his article, he compared the traditional form of producing software (dubbed the Cathedral-style model) to a way being used with "success" in several case studies of libre software development.

In the traditional model (the Cathedral), tasks and roles are clearly defined and derived from project goals specified in advance. The development takes place in a

centralized way, dividing participants into three groups: (1) people who dedicate themselves to the design of the system (the architects); (2) a group of people dedicated to the management of the project; and (3) a final group fulfilling the pertinent tasks of implementation. On the other hand, in the Bazaar model, roles are not clear and can, and do, vary during the lifetime of the project. The important property of this Bazaar model is the constant interchange of knowledge, a flow of ideas of which all take advantage, what causes the software to evolve at a faster rhythm. This interchange emerges ideally in a spontaneous way and keeps running without supervision.

Unfortunately, although there exists a clear consensus on what the Bazaar model is and what it is not, this is not a totally defined concept and therefore it is subject to subjective interpretations. There are frequently debates that arise on whether one project follows the Bazaar model or not.

While showing some similarities, it seems clear that the Bazaar development model does not propose a development methodology as extreme programming or the Rational Unified Process[3] do. The Bazaar style is formulated as a series of prescriptions – technically we could call them patterns – that should be applied in order to make a project "successful". The patterns that are to be used in a project depend on several factors, such as project size (software size and/or number of developers) or status (whether a project is in its early stages or an already consolidated project with a large user-developer base, etc.). The application of a specific pattern is not very well defined either; its use should be guided by common sense, i.e. adjusted to the circumstances of the project in question.

Through the last years, together with the evolution of libre software, the concept of the Bazaar model itself has been evolving continuously. For example, while in Raymond (1997) it was stated that projects owe their beginnings to personal initiative (generally a developer who discovered a technical necessity), time has shown how big corporations (e.g., Netscape and SUN Microsystems) changed their strategies and released the source code of some of their software products in order to benefit from the Bazaar model. Today, a growing number of libre software projects are the heirs of closed, proprietary ancestors.[4] Although it is evident that we are dealing now with different project types, it seems that there is a certain unanimity nowadays to consider both approximations as projects following the Bazaar model.

The patterns of the Bazaar model are presented with thoroughness in this book in the article by Matthias Ettrich, so it should be sufficient to give only a brief description of the most important ones:

- *Treat your users as co-developers*: That is to open the development process to the maximum, so that developers interested in the project can be integrated seamlessly. More co-developers equals the possibility (for the project) of evolving more quickly, of creating functionality at a faster pace, etc.

[3] Being strict, we should consider RUP as a meta-methodology from which software development methodologies can be derived.

[4] To name just two of the most important: OpenOffice inherited its code base from the proprietary StarOffice, and the Mozilla web browser began its existence as the Netscape browser.

- *Release early*. A first version of the software has to be released although limited in functionality as soon as there is something presentable and functional. This way, the possibility of finding co-developers interested in participating increases.
- *Frequent integration* (also known as *release often*): Taking small steps allows the project to evolve in an incremental way, at the same time enabling the parallel debugging, and frequent feedback, by co-developers and users. In addition, frequent integration avoids an expensive and tedious, and error-prone phase of integration at the end of each development cycle.[5] The release policy differs from project to project, and as shown by Ehrenkrantz (2003) it can be very complex for big projects.
- *Maintain several versions*: The goal is to not overload the more conservative users with less-tried (i.e. more buggy) versions of software. Thus, two coexisting versions exist, one which is for development (the unstable one) supplying all the latest functionality (i.e. less proved and debugged) and a production version (the stable one) for those who put emphasis on stability, rather than on functionality, of software.
- *Modularize to the maximum*: High modularization allows parallel development and code reuse. Unfortunately, code production is not as much parallelizable as deployment and debugging is.
- *Dynamic decision making structure*: Although it is generally assumed that the final decision corresponds to the developers that actually code, usually an organizational structure exists – sometimes informal, sometimes more formal – by means of which the strategic decisions are made. The organizational form depends on the project, on its technical facets (project size, base of users, number of developers,...) as well as on other causes (historical reasons, balance between interests,...). For instance, Linux counts on "a benevolent dictatorship" model, Linus Torvalds being the dictator who delegates some supervision tasks to his "lieutenants", whereas other projects trust formulas based on more democratic meritocracies (Apache, GNOME...) or comittees that represent all "families" in the development community (GCC Steering Commitee, etc.).

Until now, this article has shown a point of view of process engineering, a widely studied field in software engineering. Even so, the Bazaar development model has a component that makes it unpredictable, sometimes we could even speak about some inherent "magic" in it. "Success" seems to be very capricious: it is possible that, although using the same ingredients that made another project "successful", a new one will ultimately fail. It turns out that the Bazaar is sometimes hard to reproduce; even harder it is to predict in advance, whether the attempt is likely to succeed or is doomed to fail. In any case, all this means that we know very little of the

[5] The article by MacCormack (2001) presents overwhelming evidence for the relationship between a frequent integration-frequent release-fast feedback development model and the quality of the end-product.

Bazaar model and there has to be done a serious research effort to gain knowledge on it.

Luckily, and owed to the circumstance that the Bazaar model pushes the openness of software development to its maximum (especially using means of telematics), a vast amount of information about the projects are available – ready to be retrieved and evaluated. In addition to having access to the source code itself, we are able to obtain all the previous states of a project, e.g., from archives of older releases or from a versioning system like CVS. Further documented is a number of interactions that have taken place during software development such as postings to newsgroups and mailing lists, information from bug-tracking tools, program documentation, etc. The main goal of investigating the named resources could be expressed as to remove the "magic" from the Bazaar to make it an engineering method.

3. Quantitative analysis of source code

The availability of source code presents one of the evident advantages at the time of undertaking the study of libre software. Besides the pure code itself, a series of tools exist for supporting the coordination of the software producing development team. For example, projects usually have a central source code repository where the project's code is stored (the current version as well as all the previous ones). Whenever a developer makes a modification to his local copy of the source code, he will have to synchronize it with the repository in order to grant the other project members access to the modified code. Also, the coordination of bug tracking – notification and resolution – is usually handled by means of a (central) web application where the errors are stored and can be retrieved.

Mockus et al. (2000) presented one of the first studies on libre software projects that contained a complete description of the development process and the underlying organizational structures, including qualitative and quantitative evidences. In order to quantify aspects of the development process such as developer participation, source code size, responsibility of code, productivity, density of defects, and error resolution time intervals, the software change history (stored within the changelog of the source code repository) is scrutinized as well as information in the bug-tracking system.

As a consequence of their study, the authors put forward several hypotheses. First, the existence of a small development group (the core team) that develops most of the new functionality of a project was demonstrated. Usually this group would not be bigger than 10 to 15 persons. For such a small team, informal means of coordination are used. Depending on the size of the problem (project), a small group may not be sufficient to fulfill the task, and more developers have to be integrated into the process. In such a case, another series of coordination mechanisms are to be introduced (individual or group code ownership, etc.), even possibly dividing the project in two different (smaller) ones. A second conclusion goes that in Bazaar type projects the number of participants contributing patches (i.e. corrections of bugs) is an order of magnitude larger than the core group; and the number of those notifying bugs, again, is an order of magnitude larger than the one of

the "patchers". Third, the authors state that the time of response to user feedback is significantly shorter in Bazaar-style projects than in projects following more traditional lines. Fourth, compared to proprietary projects that had passed a comparable testing scenario, the defect density is inferior in libre software projects.

The fact that the software produced with the Bazaar model is less inclined to contain errors is also discussed by Challet and Le Du (2003). They developed a mathematical model for simulating software development and debugging processes in closed source and open source projects as well. It appears that in order to reach a density of defects as low as the one in libre software projects, closed source projects would require a high number of above-average qualified developers. The proposed model assumes that in principle users update their software with each new release. Each new release removes defects and, at the same time, introduces some new ones. Usually, more defects are removed than introduced, thus the rate of defects decreases (more or less) quickly in time. Frequent releases allow to have a process with fast feedback in the Bazaar model which can be compensated in the closed source development model only by employing developers that produce code with fewer defects.

Godfrey and Tu (2001) explored the evolution of the Linux kernel following the classical theory of software evolution (Lehman et al. 1997). The main conclusion of their article is that Linux, although being a large project (with more than two million of lines of source code), grows in a superlinear way while developers are geographically dispersed. By contrast, until now, it had been postulated that the growth of large systems tends to slow-down itself over time. When scrutinizing the tree structure of the Linux kernel more thoroughly, the authors realized that more than half of the code corresponded to device drivers. Device drivers, however, are relatively independent parts of code, hence parallel development is feasible without encountering too many problems. On the other hand, the part of code corresponding to the "heart" of the kernel is relatively small. Thus, the superlinear growth of Linux is due to its strong modularization – in technical aspects as well as in organizational ones.

In libre software the exhaustive analysis of code does not have to limit itself to the study of independent applications, as it has been the case in the previous studies. Complete collections of libre software exist that have been integrated and packaged so that the whole set works properly as an integrated environment. Such packages come together with source code and can be analyzed too. This is the case for GNU/Linux distributions that have served to popularize libre software in general, and Linux based systems in particular.

In the contributions of Wheeler (2000 and 2001) we can find a quantitative analysis of the source code[6] and the programming languages used in several versions of the Red Hat distribution. González-Barahona et al. (2001) followed with a series of articles dedicated to the Debian operating system. Using a libre software tool called

[6] These studies use the following definition for the concept of physical source line of code: 'A physical source line of code (SLOC) is a line ending in a newline or end-of-file marker, and which contains at least one non-whitespace non-comment character.' (Park, 1992).

SLOCCount[7], the studies offer some kind of "radiography" of these two popular distributions.

The studies of Red Hat and Debian show how the respective different economical nature[8] for these two distributions affects the resulting software environment. It is worth to point out that the mean package size in Debian practically remained constant throughout the last five years. The natural tendency of packages to grow (in functionality, i.e. in code size) is neutralized by the inclusion of smaller packages, whereas in Red Hat the mean package size is growing continually. The reason for this difference is that in Debian the main requirement to have a package included in the official distribution is that some volunteer exists who packs it, whereas the composition of the Red Hat product depends on the effort that a company has to dedicate to ship the next version. Thus, Red Hat has a smaller number of packages, but, in an attempt to attract the widest user base, it includes almost all the important applications of the libre software panorama, each application in turn having almost all the available features included (resulting in comparably large applications), while Debian reflects a widespread – and diverse – software "ecology".

The possibility to study the evolution of the different Debian versions over time brings to light some interesting evidence (González-Barahona et al. 2003b). It is possible to perceive the importance of the C programming language (still predominant but loosing appeal), whereas non-compiled scripting languages (Python, PHP and Perl), and the platform-independent Java, show an explosive growth. The "classical" compiled languages (Pascal, Ada, Modula etc.) are in clear recession. Also, certain characteristics of the included programming languages can be observed: languages usually have a number of source code lines per file that is more or less constant over time,[9] and object oriented languages usually have files with fewer source lines of code – one reason why a higher modularity can be supposed.

In order to finish this section dedicated to the empirical study of source code, some proposals made in Robles and González-Barahona (2003); German and Mockus (2003) should be commented. Both articles present a complete, although limited, application of a methodology for the analysis of projects embarking all the previously mentioned ones. Future investigations should include many other parameters that have not been considered yet, as for example the use of complexity metrics.

4. Analysis of human resources

The traditional development model (or Cathedral) knows anytime the human resources which it has for the accomplishment of tasks, or so it is assumed. In the Bazaar model – due to its openness and the intention to integrate anybody who

7 SLOCCount (http://www.dwheeler.com/sloccount) is a suite of programs for counting physical source lines of code (SLOC) in possibly large software systems. It can count physical SLOC for a wide number of languages. It can take a large set of files and automatically categorize their types using a number of different heuristics, and also comes with analysis tools.

8 RedHat is a commercial distribution, whereas the second one, Debian, is developed and distributed completely by volunteers.

9 With the exception of some cases such as the shell scripting language(s).

wants to collaborate – this aspect cannot be planned for. From the management point of view this raises manifold problems beginning with the impossibility to know the number of tasks that can be fulfilled to a greater uncertainty regarding estimations of development time. It is necessary, therefore, to know more about libre software developers. In this section we will see some studies, ordered chronologically, that have concentrated themselves with these questions.

The open software development taking place in a distributed way lead to the question where developers are actually located. Soon it became clear that this could be one of the possible measures of the penetration of libre software. Early studies were being focused on socio-political questions in some cases and on technological rivalry between Europe and the United States in others. Dempsey et al. (1999) was the first research in this aspect by means of the study of a Linux file repository at the Metalab archive[10].

To be included in this file repository, every package had to add a denominated Linux Software Map (LSM) file where information on the software was specified. Some data was of forced fulfillment, for instance a brief description of the program. The specification of the main author and its e-mail address where also mandatory.

From the inspection of the top-level-domain of the authors' e-mail addresses some surprising conclusions could be derived, for example the unexpectedly huge number of top-level-domains relative to Germany (second one after popular .com).

Evidently, the method used was very prone to be inaccurate – the existence of "generic" domains like .com, .net, .org or .edu that were located at the first positions should be noted, or the assumption that a piece of software had only one author. It was for reasons like these that other studies were needed.

We can find another approach to these questions in Ghosh and Prakash (2000). The primary target of this study was centered around collecting data that could support a thesis that first appeared in Ghosh (1998). Ghosh supposed the existence of a "gift economy" as the resulting product in libre software is a public good. Contributions to the development were investigated in an empirical way. The inspection process was performed automatically by a tool named CODD. CODD tracks the source code according to patterns of authorship. These data were collected and in a later step processed in order to prepare full statistics. The study included the analysis of more than 3,000 projects, containing more than 1 Gigabyte, or about 25 million lines of code.[11]

More than 12,000 different authors were found and it could also be demonstrated that the participation follows the Pareto law, also known as the 80:20 rule: 80% of the code correspond to 20% of the developers, whereas the 80% of remaining developers had a contribution of 20% of the code. Many later studies have confirmed and extended to other forms of participation the validity of this result (messages to mailing lists, bug notifications, etc.).

---

[10] Metalab changed its name recently to ibiblio. The Linux archives can be visited at http://www.ibiblio.org/pub/Linux/.

[11] In this case it is not specified if lines of binary code or lines of source code (as in SLOCCount) were computed.

In 2001 a group of students at the Technische Universität Berlin, attempting to gain better knowledge of libre software developers, carried out a combined study (WIDI 2001) comprising an Internet-based survey as well as the analysis of e-mail addresses from a libre software repository (a complete GNU/Linux distribution) with CODD as described above. A further source of information was the Debian developer database, which stores data on the volunteers who maintain the Debian distribution. The online-survey was filled up by more than 5,000 people from all continents – thus far the largest survey of libre software developers at all.[12]

Regarding the number of developers and also considering the ratio of developer per million inhabitants, the results obtained show a slight advantage of Europe in comparison with the United States. The findings also confirm those results of the study based on LSM entrances, since the positions of the countries in the four classifications correspond with it, being always headed by the United States and Germany. Another result derived from this study is the distribution of time developers spend periodically on libre software: the average occupation is about 10 hours, while the median lies between 2 and 5 hours a week (with 25% of all developers). Less than 20% of the developers answered that they would spend more than 20 hours a week developing libre software. A later study sponsored by the European Commission (FLOSS 2002) validates these results and deepens insights in other aspects.

Another empirical study that dealt with libre software developers was conducted by Krishnamurthy (2002). Therein an analysis of 100 mature projects hosted at SourceForge is made. The results show a great majority of these projects being conducted by a small number of developers (usually one). This leads the author to conclude that the term "community" is not so well suited for the world of libre software. Instead he suggests using the description of a "voluntary association". Capiluppi et al. (2003) and Healy and Schussman (2003) arrive at the same conclusions with similar studies, the first with more than 400 random selected projects from Freshmeat (a very popular libre software announcement site) and the second by means of an exhaustive study of all projects hosted at SourceForge. In any case, these three studies do not differentiate between libre software in general and software developed according to the Bazaar model as is done in this article.

One of the most recent studies on libre software developers, González-Barahona et al. (2003), demonstrates how the development teams change over time. For most of the studied projects – especially those following the Bazaar model – different generations of core developers can be identified. If this finding is correct, the often made assumption that "success" in libre software is mainly due to the leadership of a single person (known as "code god") is refuted.

5. Empirical analysis and libre software production costs

The estimation of (human and technical) resources and of the time that it takes to successfully finish a project has always been one of the big challenges in all fields of engineering. In short, resources multiplied by time can be translated into production costs. Software engineering has always been weak in giving valid estimations

[12] A further analysis of the WIDI data is currently under way.

to help manage these parameters. Even though in libre software such aspects possibly do not have that importance as in the closed business world, from an engineering point of view it would be interesting to be able to calculate these parameters. The results, then, could be compared to their counterparts of "traditional" software projects.[13]

Koch and Schneider (2000) analyzed the interactions taking place in a libre software project. Their information source were mailing lists and the versioning system repository of the GNOME "project". The most interesting aspect of this study is the economic analysis. Koch wanted to verify the validity of classical cost prediction methods (function points, COCOMO model etc.) and discovered the problems that their application entails. Although he admits that the obtained results (taken with precautions) could be partially adjusted to reality, he concludes that libre software needs its own methods and models. The known ones simply do not fit well the nature of libre software projects. Evidently, the ability to obtain many of the data related to the development of libre software allows to be optimistic.

Another of the implicit conclusions from Koch and Schneider (2000) can be extracted from the use of the Norden-Rayleigh curve. This curve is based on the original suppositions of Norden dedicated to the study of scientific projects. For Norden, this type of projects could be modeled like a series of indefinite problems (but of finite number) that require of human effort ("manpower") for their resolution. The number of persons that has to be dedicated to solve problems is approximately proportional to the number of problems still to solve in any moment in time. It also models the fact that the members of the team are constantly learning in time. Following this assumptions, at the end of the project the number of people working on the project should be small, since the number of problems is also small. It seems evident that in libre software we cannot use this model, at least not in a general way, because one of its conditions is to know the number of developers on which to base our assumptions. We also need to know the time these programmers dedicate to solve problems. Both factors are usually unknown in libre software.

Previously in this article, studies on the evolution of two GNU/Linux distributions, Debian and Red Hat, have been presented. Using models for cost calculation typically applied in "classical" software engineering, and with the supply of lines of code from libre software projects, we are in the position to give some estimations. We are going to make use of the Basic COCOMO model which was proposed at the beginning of the 1980s (Boehm 1981) and that has been extended and improved in the 1990s as COCOMO II. COCOMO allows to make an estimation of effort and development time for (proprietary) software projects carried out within the traditional waterfall model. From the calculated effort (counted in man-months) and the duration of the project (in months) the total cost of the project can be estimated.

[13] This means that the quantitative analysis of economic parameters we are considering in this section is not one about cooking-pot-markets (Ghosh 1998), which has found quite some attention among economists. The latter is concerned with aspects like the production of public goods by libre software developers. The discussion here is structured more along the lines of interest of a company that wants to produce libre software and take advantage of the Bazaar.

The following table shows the results in brief. It can be noted that the Red Hat and Debian distributions have been growing through time, the versions of the latter being the larger ones. In comparison, there are estimations that esteem Microsoft Windows 95 and Microsoft Windows 2000 having 15 MSLOC and 29 MSLOC respectively.[14] The effort required to create so much code is enormous (thousands of person-years), whereas for the estimation of the time for its accomplishment it assumes that packages are independent from each other and thus can be developed in parallel. That way, it occurs that the given time is the one that it would take to create the biggest project included in the distribution; and it is for that reason that Red Hat versions, while being smaller in size, take more time to develop than some of the Debian versions (actually being larger). Finally, the estimation of development costs for each distribution (development starting from scratch) is given.

Name	*Release*	*MSLOC*	*Effort (person-years)*	*Time (years)*	*Cost (Million USD)*
Red Hat 5.2	1998-10	12	3,216	4.93	434
Red Hat 6.0	1999-04	15	3,951	5.08	533
Red Hat 6.2	2000-03	18	4,830	5.45	652
Debian 2.0	1998-07	25	6,360	4.93	860
Red Hat 7.1	2001-04	32	8,427	6.53	1,138
Debian 2.1	1999-03	37	9,425	4.99	1,275
Red Hat 8.0	2002-09	50	13,313	7.35	1,798
Debian 2.2	2000-08	59	14,950	6.04	2,020
Debian 3.0	2002-07	105	26,835	6.81	3,625

Effort, development and total cost estimation for different Red Hat and Debian versions

Without any doubt, the numbers shown in the above table must be taken with prudent caution. The COCOMO model has been proposed for large industrial projects following (more or less) the cathedral model. Therefore, using the same formula for projects in which the development model is Bazaar-style is only the first source of mistakes, the combination of which could make the obtained estimations totally invalid. As a first albeit rough estimation, these numbers should be sufficient.

[14] These numbers have been taken from Wheeler (2000). As it may seem obvious to the reader, the source for these numbers is the own company that produces the software, Microsoft. Due to the proprietary nature of its software it is impossible to have them reviewed, even to know if only the operating system has been taken into regard or also other components as the Microsoft Office suite.

6. Looking into the future of libre software engineering

As it has been possible to see throughout this article, software engineering is centered around a series of concepts. Unfortunately, some of them are not clearly defined. Thus, language becomes a barrier to share information and facts. This is the case for concepts as the following ones (although the list should not be limited to them):

- Project: It is still not very clear how to define a libre software project. We do not know if we have to consider GNOME (a desktop environment) or one of its components like Evolution (a personal information manager) as a project. Evolution is certainly part of GNOME, but with a degree of sufficient technological and organizational autonomy so that it would make sense to consider it being itself a project.
- Success: The concept of *success* is very diffuse in libre software. In traditional software engineering we could say that a project is successful when the stated goal is reached without exceeding the estimated constraints of budget and time. In libre software similar constraints do not exist and there are no functionalities planned for. All functionalities to be included are the developed ones at the moment of the release. On the other hand, it seems that the idea of "success" that is widely used for assessing libre software projects assumes a project that has a large developer and user community – something that may depend much on the goals and aims of the project.
- Core group: The core group of developers is commonly assumed to comprise the most involved persons in the development of a project. There is, however, no exact definition of how much involvement (in terms of time, contribution of lines of code or other means etc.) is necessary to form a part of this group.

In libre software engineering there is lack of a taxonomy for projects allowing to group them according to their characteristics. This article has described thus far how the simple classification of Bazaar and Cathedral is too vague for the degree of knowledge we want to acquire about libre software and its development. Setting out to develop a better classification one does not, however, have to start from the ideas that have been exposed in Raymond (1997).

Instead, this article proposes the creation of a complete suite of tools for the analysis of libre software projects, preferably in an automatic or semiautomatic way. This suite could be used to obtain the classification that was proposed in the previous paragraph and would allow to deepen investigations of projects from different perspectives. As this article has shown, several tools exist that could be integrated into this suite.

As far as the human resources are concerned, until now all studies have been dedicated almost exclusively to the people who develop source code. The fact that there exist other important type of activities has been widely ignored. Experience shows that when working groups grow in regard to the number of participants, division of labor and specialization gain more importance. For instance, tasks of internationalization or localization of a program are usually not carried out by code de-

velopers but by others. Also, the increase in complexity of software and the growing demand for user-friendly GUIs implies that graphical designers become integrated in many projects.

Another of the aspects that researchers are beginning to approach is the analysis of social networks formed by the developers. Several studies were directed in order to expose these social networks within the world of libre software. One main goal is to identify which developers or projects are in key positions within the community (Ghosh 2003; Madey et al. 2002).

Finally, it is important that models that can cope with the inherent complexity of the development process of libre software are created. Such models would allow to understand – and to predict – certain organizational aspects. That could be a first step on the way towards the use of intelligent agents initiating the creation of desired behaviors within a development process. There are already several projects aiming at doing exactly that.[15]

7. Conclusions

In this article, the phenomenon of libre software, especially the Bazaar-style development model, has been discussed from an software engineering perspective. The characteristics of this development model that has revolutionized the world of libre software and that has sparked an intense debate on the effectiveness and efficiency of its development process (in comparison to more traditional approaches) have been presented. On the other hand, it has been pointed out that we still have a very sparse knowledge of the Bazaar model. Among others, one of its shortcomings is that "success" is hardly predictable.

Libre software offers access to the source code and other elements that serve for the purpose of developer intercommunication in the software production process. Hence, they can be studied empirically. In this article we have seen three different perspectives: one closely bound to the analysis of source code, a second one dealing with the developers of libre software and, finally, one occupied with estimations of the production costs.

The above described studies of the source code archives demonstrate that a small development group is generally responsible for a big part of the code, whereas the comparably small number of external contributions comes from a greater amount of people. On the other hand, libre software is less inclined to contain errors due to its high release rate and fast feedback cycle. Also, some projects have a growth rate never seen before in similar-sized closed software projects. That can be explained partly through the possibilities for participation offered by the Bazaar model and partly because of an efficient technical and organizational modularization.

As regards the developers of libre software, this article has shown the importance of learning more about them since, from a management point of view, human resources are one of the pillars for cost estimation and prediction of software evolution. Libre software developers are a very heterogeneous group of people, and while the great majority dedicates only a few hours a week to the development of libre

[15] Cf. Antoniades et al. (2003).

software, a small but important minority dedicates almost as much time as a part-time job would require, or more. It is important to emphasize that in great projects that follow the Bazaar model, several generations of developers have been observed, continually following each other.

Predictions of cost and effort for libre software projects fail for two reasons. First, there is the lack of information on which we can count nowadays and secondly, because assumptions of traditional models cannot be applied. We have seen how some research projects attempted to do so and ended up without satisfying results.

Finally, some of the aspects have been analyzed that have to be treated in the next future. Of greatest importance is the creation of definitions so that semantic ambiguities disappear. It should be investigated further how to better characterize projects in order to extend the present knowledge and to be able to develop new methods and models. For that aim, the creation of a suite of tools that automates as far as possible the characterization of existing projects is proposed.

Besides such technical issues, there is a clear need to enhance studies of relationships among developers through more conventional social network analysis. The emergence of new forms of labor division, e.g. in the case of translators and designers deserves further attention.

Last, but not least, the possibility of having new methods of understanding libre software by means of intelligent agents is pointed out.

The reader who desires to obtain more information and data on the advances of software engineering in the field of libre software, may visit the Libre Software Engineering website at http://libresoft.dat.escet.urjc.es.

Bibliography

Antoniades, Ioannis, Samoladas Ioannis, Stamelos Ioannis, and Bleris G.L. (2003): *Dynamical simulation models of the Open Source development process*, pending publication in: Free/Open Source Software Development, edited by Stefan Koch, Idea Inc, Vienna.

Boehm, Barry W. (1981): *Software Engineering Economics*, Prentice Hall.

Campbell-Kelly, Martin (2003): *From Airline Reservations to Sonic the Hedgehog: a History of the Software Industry*, Cambridge, MA & London: MIT Press.

Capiluppi, Andrea, Patricia Lago, and Maurizio Morisio (2003): *Evidences in the evolution of OS projects through Changelog Analyses*, online http://softeng.polito.it/andrea/publications/icse2003.pdf (28.1.2004)

Challet, Damien and Yann Le Du (2003): *Closed source versus open source in a model of software bug dynamics*, (prepublished), online http://arxiv.org/abs/cond-mat/0306511 (28.1.2004)

Dempsey, Bart J., Debra Weiss, Paul Jones and Jane Greenberg (1999): *A Quantitative Profile of a Community of Open Source Linux Developers*, online http://www.ibiblio.org/osrt/develpro.html. (28.01.2004)

Ehrenkrantz, Justin R. (2003): *Release Management Within Open Source Projects*, online http://opensource.ucc.ie/icse2003/3rd-ws-on-oss-engineering.pdf (28.01.2004)

FLOSS (2002): Rishab Aiyer Ghosh, Rüdiger Glott, and Gregorio Robles, FLOSS: *Free/Libre and Open Source Software: Survey and Study*, online http://www.flossproject.org/ (28.01.2004)

Godfrey, Michael W. and Qiang Tu (2000): E*volution in Open Source Software: A Case Study*, online http://plg.uwaterloo.ca/~migod/papers/icsm00.pdf (28.01.2004)

González-Barahona, Jesús M., Miguel A. Ortuño Pérez, Pedro de las Heras Quirós, José Centeno González, and Vicente Matellán Olivera (2001): *Counting potatoes: The size of Debian 2.2*, in: Upgrade Magazine, Vol. 2, Issue 6, December 2001, online http://upgrade-cepis.org/issues/2001/6/up2-6gonzalez.pdf (28.01.2004)

González-Barahona, Jesús M. and Gregorio Robles (2003): *Unmounting the code god assumption*, proceedings of the Fourth International Conference on Extreme Programming and Agile Processes in Software Engineering (Genova, Italy), online http://libresoft.dat.escet.urjc.es/html/downloads/xp2003-barahona-robles.pdf (28.01.2004)

González-Barahona, Jesús M., et. al (2003b): *Anatomy of two GNU/Linux distributions*, pending publication in: Free/Open Source Software Development, edited by Stefan Koch, Idea Inc, Vienna.

Germán, Daniel and Audris Mockus (2003): *Automating the Measurement of Open Source Projects*, online http://opensource.ucc.ie/icse2003/3rd-ws-on-oss-engineering.pdf (28.01.2004)

Ghosh, Rishab Aiyer (1998): *Cooking-pot markets: an economic model for "free" resources on the Internet*, online http://www.firstmonday.dk/issues/issue3_3/ghosh/ (28.01.2004)

Ghosh, Rishab Aiyer and Vipul Ved Prakash (2000): *The Orbiten Free Software Survey*, online http://www.firstmonday.dk/issues/issue5_7/ghosh/index.html (28.01.2004)

Ghosh, Rishab Aiyer (2003): *Clustering and dependencies in free/open source software development: Methodology and tools*, online http://www.firstmonday.dk/issues/issue8_4/ghosh/index.html (28.01.2004)

Helay, Kieran and Alan Schussman (2003): *The Ecology of Open Source Software Development*, online http://opensource.mit.edu/papers/healyschussman.pdf (28.01.2004)

Koch, Stefan and Georg Schneider (2000): *Results from Software Engineering Research into Open Source Development Projects Using Public Data*, online http://wwwai.wu-wien.ac.at/~koch/forschung/sw-eng/wp22.pdf (28.01.2004)

Krishnamurthy, Sandeep (2002): *Cave or Community? An Empirical Examination of 100 Mature Open Source Projects*,
online http://opensource.mit.edu/papers/krishnamurthy.pdf (28.01.2004)

Lanzara, Giovan R. and Michelle Morner (2003): *The Knowledge Ecology of Open-Source Software Projects*,
online http://opensource.mit.edu/papers/lanzaramorner.pdf (28.01.2004)

Lehman, MM, J.F. Ramil, P.D. Wernick, and D.E. Perry (1997): *Metrics and laws of software evolution – the nineties view*, proceedings of the Fourth International Software Metrics Symposium.

Levy, Steven (2001): *Hackers: Heroes of the computer Revolution*, New York, Penguin Books.

MacCormack, Alan (2001): *How Internet Companies Build Software*, in: MIT Sloan Management Review, vol. 42, no. 2 (Winter 2001), p. 75–84.

Madey, Greg, V. Freeh, and R. Tynan (2002): *The Open Source Software Development Phenomenon: An Analysis Based on Social Network Theory*,
online http://www.nd.edu/~oss/papers/amcis_oss.pdf (28.01.2004)

Mockus, Audris, Roy T. Fielding, and James D. Herbsleb (2000): *Two Case Studies of Open Source Software Development: Apache and Mozilla*,
online http://www.research.avayalabs.com/techreport/alr-2002-003-paper.pdf (28.01.2004)

Park, Robert E. et. al. (1992): *Software Size Measurement: A Framework for counting Source Statements*, Technical Report CMU/SEI-92-TR-20,
online http://www.sei.cmu.edu/publications/documents/92.reports/92.tr.020.html (28.01.2004)

Raymond, Eric S. (1997): *The Cathedral and the Bazaar, Musings on Linux and Open Source by an Accidental Revolutionary*,
http://catb.org/~esr/writings/cathedral-bazaar/ (28.01.2004)

Robles, Gregorio, Jesús González-Barahona, José Centeno-González, Vicente Matellán-Olivera and Luis Rodero-Merino (2003): *Studying the evolution of libre software projects using publicly available data*, proceedings of the 3rd Workshop on Open Source Software Engineering held at the 25th International Conference on Software Engineering,
online http://opensource.ucc.ie/icse2003/3rd-ws-on-oss-engineering.pdf (28.01.2004)

Wheeler, David A. (2000): *Estimating Linux's Size*,
online http://www.dwheeler.com/sloc (28.01.2004)

Wheeler, David A. (2001): *More than a Gigabuck: Estimating GNU/Linux's Size*,
online http://www.dwheeler.com/sloc (28.01.2004)

WIDI (2001): Gregorio Robles, Henrik Scheider, Ingo Tretkowski, and Niels Weber, *WIDI – Who Is Doing It? A research on Libre Software developers*,
online http://widi.berlios.de/paper/study.pdf (28.01.2004)