ELSEVIER

The Journal of Systems and Software 66 (2003) 77–90

The Journal of Systems and Software

www.elsevier.com/locate/jss

Controversy Corner

Open source software—an evaluation ☆

Alfonso Fuggetta *

Politecnico di Milano, Dipartimento di Elettronica e Informazione, Piazza Leonardo da Vinci 32 and CEFRIEL, Via Fucini, 2, I-20133 Milano, Italy

Abstract

The success of Linux and Apache has strengthened the opinion that the open source paradigm is one of the most promising strategies to enhance the maturity, quality, and efficiency of software development activities. This observation, however, has not been discussed in much detail and critically addressed by the software engineering community. Most of the claims associated with open source appear to be weakly motivated and articulated.

For this reason, this paper proposes some qualitative reflections and observations on the nature of open source software and on the most popular and important claims associated with the open source approach. The ultimate goal of the paper is to identify the concepts and intuitions that are really peculiar to open source, and to distinguish them from features and aspects that can be equally applied to or found in proprietary software.

1. Introduction

In the past five years, open source software has become one of the most discussed topics among software users and practitioners. The increasing interest in open source software has been motivated by at least three factors: the success of products such as Linux and Apache, which are gaining increasing shares in their own markets (operating systems and http servers); the uneasiness about the Microsoft monopoly in the software industry; and, finally, the increasingly strong opinion that "classical" approaches to software development are failing to provide a satisfactory answer to the increasing demand for effective and reliable software applications.

The interest in open source is visible at different levels and in different contexts:

- There is a very large community of *individual users* who support and promote open source. This phenomenon is particularly strong and visible in industrial and academic research, mostly in non-software areas. As a matter of fact, a large number of supporters of the open source approach are not computer scientists. This can be explained by noticing that open source has not been "created" by the computer science research community. Rather, it is the answer of users to their increasing discomfort about the cost, complexity, and constraints of many commercial products. On the other side, computer scientists (and in particular software engineers) have often not considered the open source approach as a real breakthrough in software development: they have ignored or overlooked it.
- Many *companies* are focusing their attention and effort on open source software. This is the case of important computer manufacturers such as Sun and IBM, which consider open source (or variations of this approach) as a strategic opportunity to undermine the Microsoft monopoly and to enforce the establishment of an open operational platform. Indeed, open source is also being adopted and exploited by an

☆ *Controversy corner*. It is the intention of the Journal of Systems and Software to publish, from time to time, articles cut from a different cloth. This is one such article.

The goal of CONTROVERSY CORNER is both to present information and to stimulate thought and discussion. Topics chosen for this coverage are not just traditional formal discussions of research work; they also contain ideas at the fringers of the field's "conventional wisdom".

These articles will succeed only to the extent that they stimulate not just thought, but action. If you have a strong reaction to the article that follows, either positive or negative, send it along to your editor, at card@software.org.

We will publish the best of the responses as CONTROVERY REVISITED.

* Tel.: +39-02-2399-3623; fax: +39-02-2399-3411.
E-mail address: alfonso.fuggetta@polimi.it (A. Fuggetta).
URL: http://www.cefriel.it/~alfonso.

0164-1212/03/$ - see front matter © 2002 Elsevier Science Inc. All rights reserved.
doi:10.1016/S0164-1212(02)00065-1

increasing number of companies which consider open source products such as Linux a viable and competitive alternative to proprietary solutions.
- Finally, *public institutions and governmental agencies*, especially in Europe, are increasingly interested in open source software for two main reasons. First, open source software is considered a viable strategy to counterbalance the dominance of US technology and to promote the development of a stronger European software industry. Second, the increasing reliance of governments and public administrations on software systems has generated a number of concerns about their security, safety, and trustworthiness. Moreover, public administration and governments are concerned about their dependency on specific software providers and are therefore extremely interested in identifying approaches that may help them increase their independence. In this respect, open source advocates claim that the unrestricted availability of the source code makes it possible to address these issues effectively.

The success of open source software has led a number of researchers and experts to believe that open source might really be the answer to the software crisis. Some open source advocates even say that in the future software will be open source or will not be at all. This extreme position has been partially motivated by the success of most "free internet services", where revenues are based on selling support services or advertising. Similarly, one can imagine that a software company can base its revenues on support services rather than on selling licenses or source code, even if recently this approach is being increasingly questioned and criticized (Shankland, 2001).

In general, open source software is one of the most important phenomena of the past five years. As such, it is essential to deepen our understanding of the nature of open source software and of the factors that have motivated its success. Unfortunately, most comments and observations about open source software appear to be weakly motivated or even misleading. For instance, they implicitly establish an unproved causal relationship between the software being open source and its quality; moreover, there is a confusing interplay of ethical, business, and technical motivations. As a consequence, it is difficult to really identify the peculiar and novel aspects of open source development with respect to other more traditional software development practices.

The goal of this paper is to provide a preliminary and qualitative evaluation of the open source approach, by critically discussing the claims associated with open source development. This will be accomplished by taking into account the results and findings of software engineering research, some relevant literature on the subject, a number of documents on open source development, and the experiences of the author in the field. The ultimate goal of this qualitative analysis is to provide preliminary and tentative answers to the following questions:

- What is really meant by open source?
- What is really novel and different in open source development with respect to more traditional approaches?
- What aspects of open source development can be equally applied to proprietary software development?
- Is there any causal relationship between the software being open source and its quality/effectiveness?
- What is the role of open source in business?

2. What is meant by "open source"?

The debate about the definition of open source is massive. There are two different interpretations that are currently used: "free software" and "open source".

The term "free software" originates from the GNU project and can be defined as follows (Free Software Foundation):

> Free software is a matter of liberty, not price. To understand the concept, you should think of "free" as in "free speech," not as in "free beer." Free software is a matter of the users' freedom to run, copy, distribute, study, change and improve the software. [...]
> In order for the freedoms to make changes, and to publish improved versions, to be meaningful, you must have access to the source code of the program. Therefore, accessibility of source code is a necessary condition for free software.

According to this definition, the issue is much broader than just granting unrestricted access to the source code of a software system. As discussed in more detail in Section 3, Stallman and the Free Software Foundation (FSF) assume that source cannot be "owned". In particular, their definition of free software challenges most of our assumptions about intellectual property.

The term "open source" has been coined later and, according to Stallman, it is an attempt to express the same kind of concept, but with a more prudent and palatable approach (Free Software Foundation): "The main argument for the term 'open source software' is that 'free software' makes some people uneasy". Actually, the official definition of "open source" is very close to "free software" (Opensource.Org). However, as Stallman argues (Free Software Foundation),

> The obvious meaning for "open source software" is "You can look at the source code." This is a much

> weaker criterion than "free software"; it includes free software, but also includes semi-free programs [...] and even some proprietary programs [...]. That obvious meaning for "open source" is not the meaning that its advocates intend. (Their "official" definition is much closer to "free software.") The result is that most people misunderstand what they are advocating.

Therefore, "open source" may be (and is indeed) used to mean also weaker forms of distribution of the source code. Certainly, the issue is not settled at the moment, as a number of different interpretations of the term are still commonly found [1] (e.g., Sun's *community sourcing* (Gabriel and Joy)). In this paper I will consider the terms "open source" and "free software" as synonyms. As the former has gained large popularity, it will be used throughout this paper to mean any form of software development where the source code is freely accessible to the worldwide community of users and developers. This is an *intentionally ambiguous definition* that aims at including different levels of "openness" of the source code. The reader is therefore invited to pay close attention to the different interpretations of the term "open source" in the different contexts where it is used.

As a final consideration, in order to properly frame the notion of open source, it is important to clarify the relationship between open source software and commercial software (Free Software Foundation):

> Commercial software is software being developed by a business that aims to make money from the use of the software. "Commercial" and "proprietary" are not the same thing! Most commercial software is proprietary, but there is commercial free software, and there is non-commercial non-free software.

Indeed, as the Linux case has demonstrated, open source software can also be a commercial success.

3. Ethical and social aspects

The discussion about the real definition of the term "open source" has already emphasized a first "essential" aspect: a significant component of the movement that is issuing the flag of open source development considers it more than just a technical problem (Free Software Foundation):

> Someone once said it this way: open source is a development methodology; free software is a political philosophy (or a social movement).
>
> The open source movement focuses on convincing business that it can profit by respecting the users' freedom to share and change software. We in the free software movement appreciate those efforts, but we believe that there is a more important issue at stake: all programmers [owe] an ethical obligation to respect those freedoms for other people. Profit is not wrong in itself, but it cannot justify mistreating other people.

This is the first and perhaps most important issue about the open source approach. Is software something that cannot be owned? The FSF argues that (Free Software Foundation):

> ...people have been told that natural rights for authors is the accepted and unquestioned tradition of our society. As a matter of history, the opposite is true. The idea of natural rights of authors was proposed and decisively rejected when the US Constitution was drawn up. That is why the Constitution only permits a system of copyright and does not require one; that is why it says that copyright must be temporary. It also states that the purpose of copyright is to promote progress—not to reward authors. Copyright does reward authors somewhat, and publishers more, but that is intended as a means of modifying their behavior.

I do share some of the genuine, egalitarian, and civil motivations of the FSF. The notions of copyright and intellectual property are often misused to guarantee an unrestricted control over specific technologies and products (Rifkin, 2000). Moreover, the issue of defining temporal limits to copyrights is indeed very critical. Finally, nobody should be allowed to patent generic results or general scientific findings that are part of the society's culture and heritage.

Nevertheless, there are specific contributions that it is reasonable to consider patentable. The `make` program, for instance, is the original result of the inventiveness of a researcher. The same applies to the first spreadsheet, Visicalc. Similarly, the graphical user interface was invented at Xerox and then pioneered by Apple. Should not these results be "patentable"? Unquestionably, the issue is quite critical, but it is unrealistic and improper to dismiss it by simply denying the notions of intellectual property and copyright. The analogy between "free speech" and "free (and open source) software" is misleading. Actually, "free speech" means that everybody has the right to express his own opinion. Thus, "free" refers to a *person's individual right about his/her own life and destiny*. It does not say anything about other people's obligations, except for the fact that everyone is bound to respect the universal right for free speech. The advocates of free software use the term "free" to mean that the results of a person's work should be freely available to

[1] The Appendix A contains some relevant definitions as proposed by the FSF (Free Software Foundation).

anybody else. As discussed in Section 5, a company can certainly decide to pursue a business initiative exploiting open source software, if this is perceived as a reasonable and effective strategy. However, this cannot become an imposition for all software developers.

To make the point even more specific, the Free Software Foundation lists a number of detailed reasons to explain why software should be free. In particular, it argues that intellectual property and software ownership introduces different levels of harm to the society (Free Software Foundation):

A copy of a program has nearly zero marginal cost

The first level of harm impedes the simple use of a program. A copy of a program has nearly zero marginal cost (and you can pay this cost by doing the work yourself), so in a free market, it would have nearly zero price. A license fee is a significant disincentive to use the program. If a widely-useful program is proprietary, far fewer people will use it.

Programmers also suffer psychosocial harm

Programmers also suffer psychosocial harm knowing that many users will not be allowed to use their work. This leads to an attitude of cynicism or denial.
[...]
Since the age of Reagan, the greatest scarcity in the United States is not technical innovation, but rather the willingness to work together for the public good. It makes no sense to encourage the former at the expense of the latter.

Inability to adapt programs

The second level of material harm is the inability to adapt programs. The ease of modification of software is one of its great advantages over older technology. But most commercially available software is not available for modification, even after you buy it. It is available for you to take it or leave it, as a black box—that is all.

The third level of material harm affects software development

Software development used to be an evolutionary process, where a person would take an existing program and rewrite parts of it for one new feature, and then another person would rewrite parts to add another feature; in some cases, this continued over a period of twenty years. Meanwhile, parts of the program would be "cannibalized" to form the beginnings of other programs.

Some of these issues will be discussed later in the paper (see Sections 4 and 5). As a preliminary observation, I wonder why software should be different from other goods and services:

- If software has to be free because its marginal costs is close to zero, then one may observe that many other goods have similar low marginal costs, such as books and CDs, or services such as cinema and theatre shows (what is the marginal cost of seeing a movie?). Should not they be "free" as well? This seems unrealistic. If a software developer realizes that the price of his/her product is too high (and therefore is a disincentive to use the software) then he/she will reduce it to increase sales. If there are no users who are willing to pay, it means that nobody is interested in that software or is able to pay the price. Then it is up to the developer to decide how to handle the situation.
- The issue concerning programmers' psychological harm is not just typical of software development. It is valid for any business activity. Indeed, the real issue here is how to reward programmers, and certainly it deserves a lot of attention, as for any other kind of workers in our society.
- The inability to adapt a program is certainly a critical issue that has several important facets. When a customer buys a car, he/she has the right to change it, of course. Actually, he "owns" the car. Similarly, if one buys the source code of a program (see the discussion on bespoke/custom software in Section 5.4), he/she can change the software freely. However, customers often buy the license to use software. It is like renting a car. As a consequence, the inability to modify and reuse the source code seems quite obvious. Open source advocates argue that if a customer spends some money for a software application, then he should have access to the source code that has been used to generate the executable. Again, how can this rule be imposed? What legal and rational motivations make it possible to force a developer to use a specific commercial approach?
- A problem related to the inability to see/modify the source code is the protection of consumers. The advocates of free software say that everybody has the right to see the source code of a software application, especially if it is going to be used for sensible and critical missions. Once again, the issue, as presented, is misleading. It is certainly reasonable to request that the source code of an application be accessible for inspections in order to check that the software is reliable and that it really conforms to the requirements. [2]

[2] Actually, the availability of source code is not sufficient to protect customers. As many experiences have widely demonstrated, software can be maintained effectively only if you have requirements and design documents. This and other technical issues will be discussed in more detail in Section 4.

However, this does not necessarily imply that the customer must be granted the right to freely manipulate and reuse/change/redistribute the source code. The only situation where this should be granted is when a software developer decides to stop supporting a software product or is patently unable to provide such service effectively. In this case, the protection of customers' rights requires that the producer releases the source code so that someone else can take over and provide customers' support.

- The third level of harm indicated by the Free Software Foundation concerns software development. Certainly, it is true that software is often developed incrementally. However, incremental development and improvement are not unique to software development and are pursued by many companies worldwide. Therefore, once again, one may ask why software should be granted a particular status. If the Free Software Foundation approach were applied to CD players, then one should derive that the design of these devices should be open so that each producer can reuse the innovations introduced by other producers. Notice that this issue is different from the problem of enforcing open standards (discussed later on): of course we want to be able to play the same CD on different devices. However, a producer may have invented some new technique to reduce the weight of the device or to increase the quality of the sound. Why should these innovations be made available also to the competitors "de jure"? Again, it might be the producer's choice, but certainly not an external imposition.

In conclusion, the different motivations proposed by free software advocates are not convincing and too extreme. The ultimate goal behind such claims is the desire to open the software market, nowadays dominated by few companies and in particular by Microsoft. This is certainly legitimate and appropriate. Moreover, there must not be any artificial limitation to the freedom to create and publish free/open source software. As Aigrain suggests, "the voluntary contribution of one's creation to the public domain is a right that cannot be restricted by any commercial interest" (Aigrain, 2002). Nevertheless, the concept of free software cannot be imposed as the unique choice to all the software producers who intend to distribute/sell a software product. Indeed, the notion of free software has been overloaded with too many implications, values, and meanings. The approach of the open source community is more pragmatic (Opensource.Org):

> The Open Source Initiative does not have a position on whether ideas can be owned, whether patents are good or bad, or any of the related controversies. We think the economic self-interest arguments for open source are strong enough that nobody needs to go on any moral crusades about it.
> [...]
> The Open Source Initiative is a marketing program for free software. It is a pitch for "free software" on solid pragmatic grounds rather than ideological tub-thumping. The winning substance has not changed, the losing attitude and symbolism have.

The remainder of the paper will consider the technical and economical aspects of the open source approach. A proper characterization of such more practical aspects may be useful also to rethink some of the general considerations presented in this section.

4. Technical aspects

Open source development has acquired a huge number of supporters, as it has gained the reputation of an effective technical approach to software development. Therefore, it is important to consider some of the main technical arguments presented by open source advocates.

4.1. Open source as a new way to develop software (a new process)

One of the strongest claims made by the open source community is that opening the source code enables a new and innovative approach to software development. In particular, this argument is proposed in a famous paper by Eric S. Raymond, "The cathedral and the Bazaar" (Raymond, 2000). The most important point raised by Raymond can be summarized as follows:

> Release early and often, delegate everything you can, be open to the point of promiscuity.

Raymond suggests that successful software development must be based on flexible approaches in opposition to rigid "cathedral-like approaches". Actually, this observation does not originate in the open source community. All the criticisms of the waterfall approach are based on the idea that it is often impossible to carry out a software development activity using such a rigid process. There are a number of concepts that implement this notion: rapid prototyping, incremental and evolutionary development, spiral lifecycle, rapid application development, and, recently, extreme programming and the agile software process. These approaches can be equally applied to proprietary and open source software. This observation can be further supported by considering another fragment of Raymond's paper on Linus Torvalds:

> Linus's innovation was not so much in doing quick-turnaround releases incorporating lots of user

> feedback (something like this had been Unix-world tradition for a long time), but in scaling it up to a level of intensity that matched the complexity of what he was developing. In those early times (around 1991) it was not unknown for him to release a new kernel more than once a day! Because he cultivated his base of co-developers and leveraged the Internet for collaboration harder than anyone else, this worked.

Actually, Microsoft has been applying daily builds for a long time. In particular, according to Cusumano and Selby's study of Microsoft development processes, the focus of that company is on features and product vision (Cusumano and Selby, 1995). Features are changed very frequently and adapted to take into account new ideas and customers' feedback. Microsoft daily builds and feature orientation are used exactly to address the issues raised by Raymond. I hereby include an excerpt from Cusumano and Selby's book; it is the description of the Microsoft process according to a Microsoft manager:

> This is the daily build process. You get a phone call saying, "Okay, we are ready to take your check-in." You check it in, and then you send mail to an alias called NT-build that describes exactly the operations to do to get that into the source pool. [...] Now go to this directory and link this, and you will get a new kernel, a new whatever.

We cannot say that this "bazaar" effect is made possible or "caused by" the software being open source. These two notions are orthogonal.

Raymond suggests that another essential success factor of Linux was the availability of the Internet as an extraordinary cooperation means. However, the same style of cooperation can be (and is, indeed) pursued for proprietary software, which is in many situation developed by different organizations or units often distributed in different countries (e.g., the software factories in Bangalore, India, which work for US and European companies).

Another important point raised by Raymond concerns motivation:

> Linus was keeping his hacker/users constantly stimulated and rewarded—stimulated by the prospect of having an ego-satisfying piece of the action, rewarded by the sight of constant (even daily) improvement in their work.

Certainly, people working in an open source project are driven by strong motivational factors. So most of the success of products such as Linux is to be credited to the commitment and ability of the people who participate in their development. But can one claim that the only way to motivate people is by opening the software code? Is not it true that people can be motivated also with money, personal success, company loyalty, and other factors not necessarily related to open source?

Finally, another important remark from Raymond's paper concerns management:

> There is another kind of skill not normally associated with software development which I think is as important as design cleverness to bazaar projects—and it may be more important. A bazaar project coordinator or leader must have good people and communications skills.

Actually, nowadays any management approach or method, in any field of engineering and business, stresses the importance of these skills. Again, the two issues are orthogonal: good companies (in any domain) do consider management skills; bad companies do not.

More in general, the critical analysis of Raymond's arguments suggests two considerations. First, there is no factual argument to support the claim that the Bazaar-style of development is "the" best approach for any software development project. Would it work for—say—an avionics system? Second, it is not true that open source is the only way to enable a Bazaar-style development. Certainly, open source is a strong motivational and catalyzing factor, but there is no evidence that it directly causes or that it necessarily and uniquely implies all the benefits of the Bazaar-style process.

4.2. Open source, requirements, architecture, and distributed development

In his paper, Raymond includes the following comments:

> Perhaps in the end the open-source culture will triumph not because cooperation is morally right or software "hoarding" is morally wrong (assuming you believe the latter, which neither Linus nor I do), but simply because the closed-source world cannot win an evolutionary arms race with open-source communities that can put orders of magnitude more skilled time into a problem.
>
> It may well turn out that one of the most important effects of open source's success will be to teach us that play is the most economically efficient mode of creative work.

These comments have two different implications. First, they once again stress the importance of pursuing a flexible, democratic, and creative development process. Second, they emphasize the fact that open source has

involved a large number of people. Indeed, there has been a very important and often overlooked factor that has made possible to develop Linux with this distributed workforce: *people knew what they were doing and how to do it*.

- Software engineering research and practice has demonstrated that an important and critical factor in software development is *requirements*. You have to know what you want to do. The requirement elicitation and specification activity (or problem definition) is extremely difficult and critical. The developer has to correctly understand users' requirements, which often derive from unknown or complex application domains (e.g., stock trading or air traffic control). Moreover, different developers can effectively cooperate if they have a common vision of what they are going to build. In general, requirements elicitation and sharing is an essential step of any development activity, especially if it has to be accomplished by a distributed team of developers.
  The development of Linux (as indicated by Raymond in his paper) was possible because the community of developers had a wide and shared knowledge of Unix, i.e., they had a "living" piece of software that embodied all the requirements for the new system being implemented. Anybody had "free" access to the definition of concepts such as "process fork and exec", "daemons", "file system operations", and "interrupt handling".
- Similarly, as Raymond states, the *basic design* of Linux was accomplished by Torvalds, reusing concepts from Unix. As a consequence, the software architecture of the system was well defined and well known to anybody else who wanted to cooperate in the project. Finally, Raymond says that he "restrained [his] tendency to be clever".

These observations confirm some basic notions of software engineering: *cooperation is possible if developers share the knowledge about requirements and architecture*. This also might explain why most open source software deals with system software: operating systems, compilers, web servers. Indeed, the community of developers can more easily share information about this kind of products. Can we assert that the same approach works equally well for business software or for innovative research projects? At the moment, there is no evidence that this is the case. Similarly, is it necessary to make the code "open source" in order to pursue distributed development activities? Once again they are orthogonal issues: distributed development is not an exclusive property of open source software.

Notice, that some widely-cited open source products have gone open source *after they have been completed* (or once they have reached a significant level of development). Zope is an example of a product that adopted this strategy. Therefore, it is improper to relate the quality of such products to the open source development process.

4.3. Open source vs. open standards

A number of researchers and practitioners claim that open source is essential to enable interoperability among different products in an open market. The classical examples cited to support this claims are the Internet and the GSM cellular system. This observation is again misleading. *Interoperability is achieved through open standards*. This does not necessarily imply that the software be open source. Standards usually define interfaces. Someone can implement the interfaces through open source software. Others may decide to have their own proprietary software to provide the same set of features.

Open source advocates claim that open source software is the only way to guarantee that standards are really open and are not jeopardized by companies who wants to establish their own monopoly. I argue that the only way to solve this problem is by urging antitrust authorities to prohibit any kind of incorrect practice.

4.4. Open source software is more reliable

Open source advocates argue that (Opensource.Org)

> The open source model also means increased security; because code is in the public view it will be exposed to extreme scrutiny, with problems being found and fixed instead of being kept secret until the wrong person discovers them.
> [...]
> Gerald P. Weinberg once famously observed that, "If builders built houses the way programmers built programs, the first woodpecker to come along would destroy civilization." He was right. Up to now, the reliability of most software has been atrociously bad.

Again, these statements are misleading and also a bit naïve.

1. In general, the underlying vision is that software developers are bad guys who try to hide software bugs. Therefore, by reading the source code, users can find problems and fix them. First, it is naïve to imagine that software developers are really interested and willing to keep bugs secret. Second, users are able to find and fix the code only if they are good programmers and if they know the code very well. This was true for most initial users of Linux and Apache, since the majority of those users were system

developers themselves. For most end-users, the availability of source code is absolutely insignificant: they would not know what to do with it. Even good programmers will find difficult or even impossible to debug a large program or a program that does not have design documents describing its structure and operations. Many reported experiences about the maintenance of large software systems have clearly demonstrated that source code is often not enough. Of course, you need it to fix the code once you have identified the bug. However, the difficult part is really to identify the bug and to devise the way to remove it. Indeed, there are a large number of researchers and companies working on "program comprehension" and "reverse engineering", i.e., tools, techniques, and methods to help derive from the code useful high-level information to make it possible its evolution.

2. People keep complaining about the low reliability of software systems. Weinberg's statement above is a typical example. However, these complaints are often quite superficial. Certainly, software programs are far from being perfect. However, there are some important observations in this respect:
   - Several years ago Fred Brooks described the nature of software, pointing out its essential difficulties. He observed that software, compared to other artifacts, has a very high number of states, and it is therefore extremely difficult to manage and control it. Can we really compare software and houses, as Weinberg does?
   - The growth in complexity of software systems along the past years is absolutely amazing. For instance, twenty years ago there were no word processors, spreadsheets, graphical tools, and multimedia authoring systems. Nowadays, these tools are used by millions of users worldwide. There has never been another industrial sector that has experienced the same successful growth (except for semiconductors). Would that be possible if the reliability of these tools were "atrociously bad"?
   - Most modern and complex services and products are run through software. For example, airplanes and traffic control systems are run by very complex and safety-critical software. Millions of lives are everyday under the responsibilities of these systems. Still, the number of casualties or severe accidents due to software problems is very low (if not zero). On the contrary, how many lives are lost everyday for car accidents caused by defective tires or mechanical problems? Similarly, worldwide financial services rely on software. They provide non-stop services to millions of users every single day. Can we really say that the reliability of these systems is low? On what basis?

     We should have real scientific argument to judge the reliability of software systems and to compare it with the reliability of other goods.
3. Software inspections have been invented more than 20 years ago. Software engineers use them in many different settings. For instance, AT&T performs a huge number of inspections every year on the software code running the telephone network. The novel aspect found in open source development is that a potentially larger number of users can check the code. However, the principle is not new. Moreover, the ability of seeing the code does not necessarily require that the code be open source. It would be sufficient to make it "accessible" to users.

Notice that *I do not want by any means to claim that software is perfect and that does not need to be improved.*

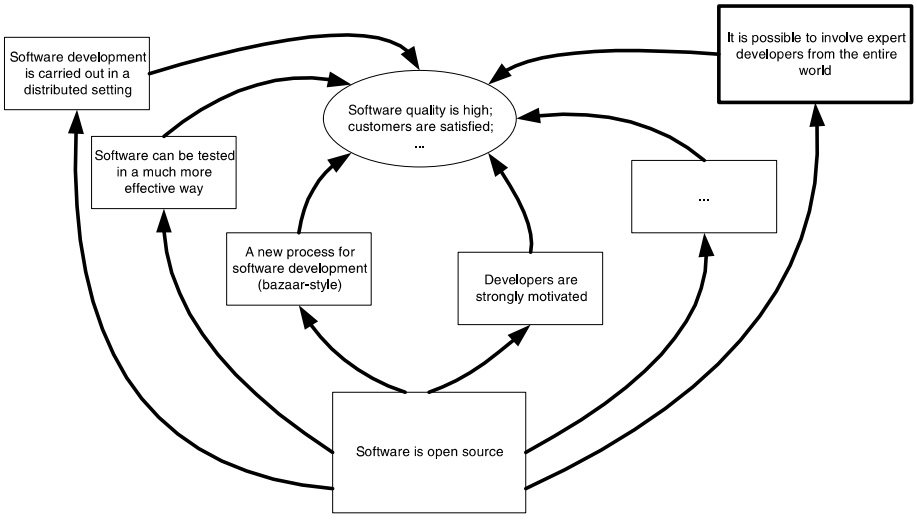

Fig. 1. Cause-effect diagram.

I am simply stating that many claims about the low reliability of software are not based on scientific evidence and are in most cases misleading.

4.5. A final remark

In general, it is difficult to affirm that there is a causal relationship between the software being open source and its effectiveness, quality, and value. *The success of products such as Linux and Apache are due to a combination of effects and there is no evidence that these effects could be obtained "iff" the software is open source*. Fig. 1 illustrates in more detail the situation. Open source is a factor that is supposed to enable some beneficial phenomena (such as "software can be tested in a much more effective way"). In turn, these phenomena cause software to be good. This cause-effect chain is not scientifically proved. In particular, it is not proved that phenomenon such as "software can be tested in a much more effective way" can be enabled if and only if software is open source. The discussion of the previous paragraphs aimed at refuting this assertion.

Certainly, one may argue that open source acts as a very good *catalyst* that enables the "reaction" through which all these different factors are mixed together. Moreover, the cultural mixing effect of bazaar projects might have an impact on quality. This is certainly a possibility that must be better studied and understood.

5. Economic/business aspects

The previous two sections of this paper have discussed ethical and technical issues related to open source software. In this section, I want to discuss some issues related to economic and business aspects. The starting point of this analysis is Raymond's second seminal paper on open source, titled "The magic cauldron" (Raymond, 1999). Other relevant literature and sources will also be cited and discussed.

5.1. Selling services vs. products

Raymond suggests that the evolution of the software market will eventually lead to a situation where "we require a price structure founded on service contracts, subscriptions, and a continuous exchange of value between vendor and customer. [...] we can predict that this is the sort of price structure most of a mature software industry will ultimately follow."

Of course, there is nothing wrong in selling services. The service-based business model is not new or unreasonable. However, it is questionable to link the price structure to the software being open source. Application service providing is a business model based on selling services and does not require code to be open source. *Therefore, the two issues (price structure and open source) are fairly orthogonal.*

In general, there might certainly be a shift from buying licences for software to renting it. The critical factor that will push customers to use either approach will be the evaluation of the total cost of ownership. Buying new versions of a software package is not necessarily more expensive than paying a yearly subscription to a maintenance service. The real difference is that if the source code is open (or owned by the user), then it is possible to change the company in charge of maintaining it, as discussed in more detail later on.

5.2. Open source as a commercial weapon/strategy

Opening the source code of software has proved to be a powerful commercial weapon/strategy. However, in most cases the key issue is that open source software is "free" as in "free beer", not that the source code is open. Let us consider some typical examples.

The browser war

Netscape has decided to open the source code of its browser to counterbalance Microsoft move to distribute Explorer "for free" (as in "free beer"). Actually, the vast majority of end-users do not care about the source code. They have been moving to Explorer because it is free, bundled with Windows, faster and better than Netscape. Netscape decision to open the source code has had two effects. The first one was to make the Netscape browser "free" (as in "free beer"). This decision was by far the most relevant one, as it addressed the key challenge posed by Explorer, i.e., cost. The second one is to reinforce the role of the Netscape browser as the "open" alternative to Explorer, i.e., it was a "selling pitch" for a specific audience (people who "do not like" Microsoft). For both companies, however, the browser is a commercial weapon to contend a key market: server software. The fact that Netscape is open source is marginal. The real issue is that by keeping a large share of the browser market, Netscape was able to sustain its sales of enterprise software and services (Cusumano and Yoffie, 1998).

Sun and Star Office

Sun is actively promoting Star Office. Star Office is not considered a true open source software, but it is free, at least partially open, and it is considered a "free" alternative to Microsoft Office. Indeed, this is the point. Sun uses "free" software as a commercial weapon to promote the diffusion of Sun workstations as an office automation alternative to the Wintel

platform. Sun revenues are from hardware, not software. Sun investments and costs to distribute Star Office are marginal with respect to the revenues made with hardware. Star Office is used to capture increasing shares in the hardware market. In practice, the key issue is again "free" as in "free beer", not open source. Open source is an accidental factor in a much broader war.

IBM, Sun, and Apache

IBM and Sun are actively supporting the development of Apache. Apache is a very good product and, being "free", it is a great option for many customers. Of course, it needs to be supported. This cost is sustained by the above mentioned companies whose interest is very clear: by defending Apache, they defend their hardware product lines. If Microsoft http server becomes the leader or the only player, nobody would buy non-Intel servers anymore.

In general, open source is often used as a commercial strategy/weapon. In most cases, the real "essential" factor is that open source software is "free" as in "free beer". In some situations, the investments in software development are covered by companies who are interested in protecting or approaching a market. For other companies (see for instance the Zope system cited in a previous section), open source is a true market strategy. In general, open source development is made possible by someone who decides to invest as he/she anticipates revenues that are not directly derived from selling licenses or source code. This is legitimate, but there is no evidence that this is a general, universal, or unique strategy for any software-based business.

5.3. Who pays the cost of software development?

Software does cost and it costs a lot. Open source software costs as well. The issue is therefore who is going to pay for open source software development. This observation is being considered very carefully even by open source advocates (Shankland, 2001).

Two possible answers have been discussed in a previous section. First, open source development may be an investment that pays off through selling services. Second, open source software may be funded by someone who is interested in protecting or opening a market.

Stallman provides some additional answers:

If we eliminate intellectual property as a means of encouraging people to develop software, at first less software will be developed, but that software will be more useful.
[...]
The question, "How can we pay programmers?", becomes an easier question when we realize that it is not a matter of paying them a fortune. A mere living is easier to raise.
[...]
Institutions that pay programmers do not have to be software houses. Many other institutions already exist which can do this.
Hardware manufacturers find it essential to support software development even if they cannot control the use of the software.
[...]
Universities conduct many programming projects. Today, they often sell the results, but in the 1970s, they did not. Is there any doubt that universities would develop free software if they were not allowed to sell software? These projects could be supported by the same government contracts and grants which now support proprietary software development.
[...]
Programmers writing free software can make their living by selling services related to the software.
[...]
New institutions such as the Free Software Foundation can also fund programmers.
[...]
The FSF is a charity, and its income is spent on hiring as many programmers as possible. If it had been set up as a business, distributing the same free software to the public for the same fee, it would now provide a very good living for its founder.
Because the Foundation is a charity, programmers often work for the Foundation for half of what they could make elsewhere. They do this because we are free of bureaucracy, and because they feel satisfaction in knowing that their work will not be obstructed from use. Most of all, they do it because programming is fun. In addition, volunteers have written many useful programs for us. (Recently even technical writers have begun to volunteer.)
This confirms that programming is among the most fascinating of all fields, along with music and art. We do not have to fear that no one will want to program.

The points made by Stallman are quite naïve. We cannot postulate that all software systems are developed just because "programming is fun". Nor can we imagine that all programmers are happy to work for "a mere living". Also, funding open source software development through public money is not a viable general strategy. Why software should be funded that way while other products are not? Finally, Stallman seems to overlook the amount of software that has to be developed and the related market size. According to the 2001

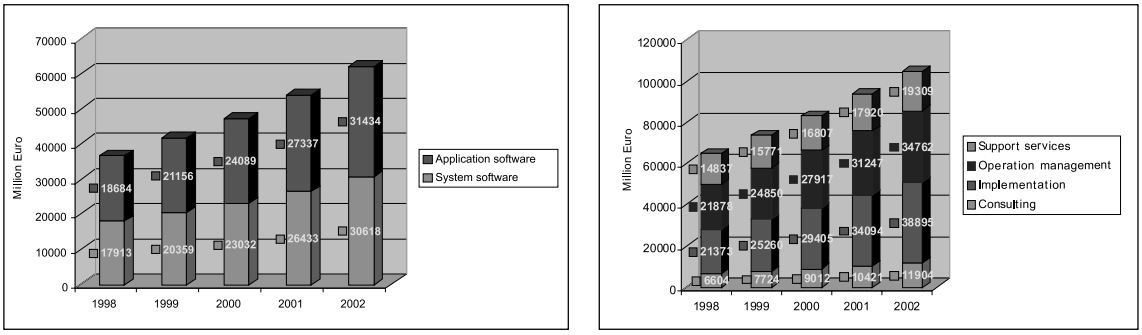

Fig. 2. Software packages market (left) and software-related services (right) in the EU (EITO, 2001).

EITO Report, in recent years the European Software market has been characterized by the trends and figures illustrated in the charts of Fig. 2. In particular, as indicated in the chart on the right, in Europe the cost of implementing custom software (not considering all the other related services) is expected to reach 34 billion Euros in year 2001. Open source advocates would probably argue that this huge amount of money is (at least partially) used very badly, since similar software systems are often implemented more than once; thus, they would argue that these actual costs are much higher than what it would be possible to achieve if a wiser strategy is adopted. Also, there are large expenditures in package licensing (see Fig. 2, left chart). Therefore, the "real and ideal" total funding needed to develop the "required" software may be much less than what we spend today. Still, it is essential to note that there is no evidence to support Stallman's claim according to which the approaches he proposes to funding are enough to cover all of the "real" needs. Moreover, there might be applications areas or domains where it is difficult or unrealistic to use any of the approaches proposed by Stallman. In general, Stallman's proposals are not harmful if they are considered possible options to explore; they may turn out to be quite critical and dangerous if imposed by forcing all software to be open source (as the Free Software Foundation advocates).

Nevertheless, there are particular market situations where open source is probably the only viable solution to support successful and effective software development. Typical examples are research communities that need specific software products to support their research work. For instance, in Astrophysics there are a large number of researchers developing open source software to support research activities. In this case, the expertise needed to develop this critical and specific kind of software, the cost of developing it, and the limited market base for the resulting products make it difficult to exploit a traditional approach to software development. Typically, in these situations software is cooperatively developed by a number of researchers worldwide. They know and share requirements very effectively, and are able to combine and integrate different software components to build complex and sophisticated computational systems. This is a situation where open source software appears to be the only viable solution. In this case, the costs of software development are supported by research institutions and are combined to achieve a high degree of synergy and effectiveness.

5.4. Open source and the protection of customers

There is an important issue that needs to be discussed in detail: open source software is considered a way to protect customers and, in particular, public administrations, government agencies, and institutions. Open source software is considered particularly attractive because it makes it possible to achieve the following goals:

- Software can be inspected to check whether it is compliant to security and safety requirements.
- Software maintenance can be managed in a much more effective way, since a customer is free to change the company maintaining the software.

I already noticed that the first point can be addressed by simply forcing companies to make software code visible to customers. This does not necessarily mean that a software producer has to make his/her software free/open. Furthermore, I claim that the real issue underlying most open source advocates' concerns is not the unavailability of source code: it is the *quality and structure of the procurement process used to purchase software*.

Customers usually purchase two kinds of software products: packages and bespoke (or custom) software. Packages (e.g., MS Office, Oracle, and SAP) are usually purchased by acquiring licenses. In this case, it is indeed impossible to change the company in charge of maintaining the source code. However, most of the customers' real problems are related to bespoke software. A typical example of bespoke software is the information

system developed to automate specific processes in a government office (e.g., driving license distribution). In this kind of situations, the software system is developed by a software company for that specific customer who pays the entire cost of developing it. *A wise and obvious way of procuring this kind of software is based on the acquisition of the ownership of the source code, since the customer pays for its development*. Unfortunately, in most situations customers simply buy the executables. This is the real source of the problem: if a customer pays the price of bespoke software development without acquiring the ownership of the source code, then, of course, he/she will be unable to change the company in charge of maintaining the system (or to accomplish any activity that requires the availability of the source code).

In this situation, the easiest and most obvious solution is to correct the procurement process. Customers do have the "power" to impose this change: for instance, in a public bid for software development activities, a public administration can and should simply state as a contractual clause that the ownership of the developed software will be transferred to the customer at the end of the project. This approach has been pursued by Italian public administrations during the past decade. When a public administration purchases (typically through a public call) some bespoke software, it becomes owner of the software itself. This makes it possible for the public administration to change the software autonomously or to use maintenance services offered by different companies. Moreover, in Italy there is a law stating that a public body or agency who owns the source code of a software system is allowed to redistribute it for free to any other public administration that can customize it and adapt it to its own needs and requirements (Gazzetta Ufficiale della Repubblica Italiana, 2000). For instance, in Italy recently the Department of Justice has installed a customized version of some bespoke software originally acquired by the Department of Treasury. This is a quite obvious consequence of an effective procurement process, which transfers to the customer (in this case, the government) the property of the bespoke software being developed.

This approach to bespoke software procurement does not require any discussion on software being open or free. It achieves the same effect by exploiting well accepted and even obvious market rules:

- "If I pay for the development of some bespoke software, then the source code is mine."
- "If the source code is mine, I can do whatever I want with it."

Notice that bespoke software development constitutes a very large portion of the software expenditures in both private companies and public administrations. Therefore, by improving the procurement process for this kind of software we would be able to solve a very large number of situations that are considered very critical by free software advocates.

6. Conclusions

Open source software is an important phenomenon that needs to deeply studied and understood. Unfortunately, the discussion about open source software has been carried out as a religious war by some, and simply ignored by others. This paper is an attempt to provide a critical and scientific (even if qualitative and preliminary) evaluation of the approach. The main conclusions that I have drawn can be summarized as follows:

- Most claims associated to open source and the related development process do apply also to proprietary software.
- It is not proved that open source uniquely and necessarily causes software to be better, more reliable, or cheaper to develop.
- Many economic and business issues are not related to software being "open" or "closed".

Still, open source software has some distinctive features and characteristics that deserve to be studied and understood so that we can exploit them to further increase the quality of software and of software development processes. I argue that these distinctive features can be summarized as follows:

Motivation

The open source approach is certainly a very effective and rewarding way to involve people in a software development project. It may be considered a strong "catalyst" to promote effective development practices.

Commercial weapon to defeat competitors or to protect a market

Open source is used as a commercial weapon to attack competitors (e.g., as in the Star Office case). In most cases, however, the real important factor is that open source software is "free" (as in "free beer").

Commercial strategy to create a community of users

Some software developers are exploiting open source to create a community of users (e.g., as in the Zope case). Thus, open source is a commercial strategy to acquire new market shares.

Closed group of users, specific markets where traditional business models do not work

Open source is a form of development that is successful whenever there are restricted or limited communities of users where traditional market strategies do not work. Of course, this requires the autonomous investment of the community or some other source of funding (typically, the government).

Effective dissemination means

Open source is a very powerful means to disseminate innovation and research results.

Of course, these conclusions are preliminary and based on qualitative observations. It is indeed important to further study these issues using empirical studies and other scientific research approaches in order to deepen our knowledge of open source software. This is essential to properly handle expectations (we must know what open source can really deliver), to exploit innovative and distinctive open source practices in the software industry, and, in perspective, to increase the quality of the software products used in our society (Rosenberg, 2000; Rubini).

Acknowledgement

This paper has been written while visiting at the University of California, Irvine (Summer 2001).

Appendix A. Definitions related to open source

These are some basic definitions taken from (Free Software Foundation):

Free software

Free software is software that comes with permission for anyone to use, copy, and distribute, either verbatim or with modifications, either gratis or for a fee. In particular, this means that source code must be available. ''If it is not source, it is not software.''

Open Source software

The term ''open source'' software is used by some people to mean more or less the same thing as free software.

Public domain software

Public domain software is software that is not copyrighted. It is a special case of non-copylefted free software, which means that some copies or modified versions may not be free at all.

Copylefted software

Copylefted software is free software whose distribution terms do not let redistributors add any additional restrictions when they redistribute or modify the software. This means that every copy of the software, even if it has been modified, must be free software.

Non-copylefted free software

Non-copylefted free software comes from the author with permission to redistribute and modify, and also to add additional restrictions to it.

If a program is free but not copylefted, then some copies or modified versions may not be free at all. A software company can compile the program, with or without modifications, and distribute the executable file as a proprietary software product.

GPL-covered software

The GNU GPL (General Public License) is one specific set of distribution terms for copylefting a program. The GNU Project uses it as the distribution terms for most GNU software.

Semi-free software

Semi-free software is software that is not free, but comes with permission for individuals to use, copy, distribute, and modify (including distribution of modified versions) for non-profit purposes. PGP is an example of a semi-free program.

Proprietary software

Proprietary software is software that is not free or semi-free. Its use, redistribution or modification is prohibited, or requires you to ask for permission, or is restricted so much that you effectively cannot do it freely.

Freeware

The term ''freeware'' has no clear accepted definition, but it is commonly used for packages which permit redistribution but not modification (and their source code is not available).

Shareware

Shareware is software that comes with permission for people to redistribute copies, but says that

anyone who continues to use a copy is required to pay a license fee.

Commercial software

For the GNU Project, the emphasis is in the other order: the important thing is that GNU Ada is free software; whether it is commercial is not a crucial question. However, the additional development of GNU Ada that results from its being commercial it is definitely beneficial.

Please help spread the awareness that commercial free software is possible. You can do this by making an effort not to say "commercial" when you mean "proprietary".

References

Aigrain, P., 2002. Positive intellectual rights and information exchanges. In: Century, M. (Ed.), CODE, MIT Press, in press.

Cusumano, M.A., Yoffie, D.B. 1998. Competing on the Internet Time. Touchstone Book (Simon & Schuster).

Cusumano, M.A., Selby, R.W., 1995. Microsoft secrets. The Free Press.

EITO 2001: European Information Technology Observatory 2001. EITO, 2001.

Free Software Foundation (FSF). "The Free Software Definition". Available from <http://www.gnu.org/philosophy/free-sw.html>.

Free Software Foundation (FSF). "Why 'Free software' is better than 'Open Source"'. Available from <http://www.gnu.org/philosophy/free-software-for-freedom.html>.

Free Software Foundation (FSF). "Categories of Free and Non-Free Software". Available from <http://www.gnu.org/philosophy/categories.html>.

Free Software Foundation (FSF). "Interview: Richard M. Stallman". Available from <http://www.gnu.org/philosophy/luispo-rms-interview.html>.

Free Software Foundation (FSF). "The GNU Manifesto". Available from <http://www.gnu.org/gnu/manifesto.html>.

Free Software Foundation (FSF). "Free Software is More Reliable!" Available from <http://www.gnu.org/software/reliability.html>.

Free Software Foundation (FSF). "Selling Free Software". Available from <http://www.gnu.org/philosophy/selling.html>.

Free Software Foundation (FSF). "Why Software Should Not Have Owners". Available from <http://www.gnu.org/philosophy/why-free.html>.

Free Software Foundation (FSF). "Why Software Should Be Free". Available from <http://www.gnu.org/philosophy/shouldbefree.html>.

Free Software Foundation (FSF). "Some Confusing or Loaded Words and Phrases that are Worth Avoiding". Available from <http://www.gnu.org/philosophy/words-to-avoid.html>.

Gabriel, R.P., Joy, W.N., Sun Community Source License Principles". Available from <http://www.sun.com/981208/scsl/principles.html>.

Gazzetta Ufficiale della Repubblica Italiana. "Disposizioni per la delegificazione di norme e per la semplificazione di procedimenti amministrativi—Legge di semplificazione 1999". Supplemento ordinario alla "Gazzetta Ufficiale, n. 296 del 20 Dicembre 2000", Serie Generale, Parte Prima.

Opensource.Org. "The Open Source Definition". Version 1.8. Available from <http://www.opensource.org/docs/definition.html>.

Opensource.Org. "Advocacy, Frequently Asked Questions". Available from <http://www.opensource.org/advocacy/faq.html>.

Opensource.Org. "Advocacy, The Open Source Case for Business". Available from <http://www.opensource.org/advocacy/case_for_business.html>.

Opensource.Org. "Advocacy, The Open Source Case for Customers". Available from <http://www.opensource.org/advocacy/case_for_customers.html>.

Opensource.Org. "Advocacy, The Open Source Case for Hackers". Available from <http://www.opensource.org/advocacy/case_for_hackers.html>.

Opensource.Org. "Advocacy, Software Secrets: Do They Help or Hurt?" Available from <http://www.opensource.org/advocacy/secrets.html>.

Raymond, E.S., 1999. "The Magic Cauldron". June 1999. Available from <http://www.tuxedo.org/~esr/writings/magic-cauldron/>.

Raymond, E.S., "The Cathedral and the Bazar". 2000. Available from <http://tuxedo.org/~esr/writings/cathedral-bazaar/>.

Rifkin, J., 2000. The age of access. Penguin Putnam.

Rosenberg, D.K., 2000. Open Source, The Unauthorized White Papers. IDG Books Worldwide, Inc.

Rubini, A., "Software Libre and Commercial Viability". Free Software Foundation (FSF). Available from <http://www.gnu. org/philosophy/software-libre-commercial-viability.html>.

Shankland, S., 2001. "Is open source fading away?" ZDNet Tech Update, November 20th, 2001. Available from <http://techupdate.zdnet.com/techupdate/stories/main/0,14179,5099830,00.html>.

Alfonso Fuggetta is a full professor of Software Engineering at Politecnico di Milano. He is also deputy director of CEFRIEL, a research and education center established in 1988 by Politecnico di Milano, University of Milano, the Regional Council of Lombardy and several ICT companies.

Alfonso Fuggetta's interests are in software process, distributed architectures and internet-wide infrastructures, e-government, and economic and organizational issues related to the adoption of ICT systems and technologies.

Alfonso Fuggetta has been a visiting professor at the Norwegian Institute of Science and Technology (Trondheim), at the University of Colorado at Boulder (USA), and at the University of California, Irvine (USA). He has been program co-chair of the 1997 edition of the International Conference on Software Engineering (Boston, USA), and is a member of the editorial boards of ACM Transactions on Software Engineering and Methodology, Software process—Improvement and Practice, and Automated Software Engineering.