Free Software / Open Source: Information Society Opportunities for Europe?

Working group on Libre Software[1]

April 2000
Version 1.2 (work in progress)

[1]The working group on Libre Software was created at the initiative of the Information Society Directorate General of the European Commission. The members of the group are Carlo Daffara (Conecta Telematica, Italy), Jesús M. González-Barahona (Universidad Rey Juan Carlos, Spain), Edmund Humenberger (G.A.M.S., Austria), Werner Koch (German Unix User Group, Germany), Bernard Lang (INRIA, France), and Ben Laurie. Some representatives of the European Commission services have also contributed with their input to this paper. They are Philippe Aigrain (Information Society Directorate General, Unit E2), Laurent Cabirol (Information Society Directorate General, Unit 1), and Michel Lacroix (Information Society Directorate General, Unit E2). This paper was edited by Carlo Daffara and Jesús M. González-Barahona. The current version of this paper is available at http://eu.conecta.it, together with a mail list for discussion about it.

Abstract

Open source software is becoming the most interesting ‘new’ phenomenon of the entire information technology landscape, generating a level of interest similar to that of the first moments of the Internet. However, as we will show in this document, the open source software phenomenon is not historically new, although in recent years it has reached a critical mass, which has allowed it to enter the mainstream software market.

The impact of open source technology is expected to be quite noticeable in the software industry, and in society as a whole. It allows for novel development models, which have already been demonstrated to be especially well suited to efficiently take advantage of the work of developers spread across all corners of the planet. It also enables completely new business models, which are shaping a network of groups and companies based on open source software development. And it has, in general, a very positive impact as an enabler for the creation of new markets and business opportunities.

Despite these facts, many people think that the open source movement is merely another temporary fashion in the software industry. On the contrary, many other believe that changes caused by open source will be so deep that they will completely shape the software industry of the first decade of the 21st centuy. This document tries to provide some facts, opinions and references, so that the reader can decide if all of this is just nonsense, or if it deserves more and better study and consideration.

This is a living document, updated from time to time. If you want to get the latest 'stable' release, you can download it.

The current version of this article is available at `http://eu.conecta.it/`

Licence

Start of OR Magazine License

This document may be freely read, stored, reproduced, disseminated, translated or quoted by any means and on any medium provided the following conditions are met:

1. **Every reader or user of this document acknowledges that she is aware that no guarantee is given regarding its contents, on any account, and specifically concerning veracity, accuracy and fitness for any purpose.**
2. **No modification is made other than cosmetic, change of representation format, translation, correction of obvious syntactic errors, or as permitted by the clauses below.**
3. **Comments and other additions may be inserted, provided they clearly appear as such; translations or fragments must clearly refer to an original complete version, preferably one that is easily accessed whenever possible.**
4. **Translations, comments and other additions or modifications must be dated and their author(s) must be identifiable (possibly via an alias).**
5. **This licence is preserved and applies to the whole document with modifications and additions (except for brief quotes), independently of the representation format.**
6. **Any reference to the "official version", "original version" or "how to obtain original versions" of the document is preserved verbatim. Any copyright notice in the document is preserved verbatim. Also, the title and author(s) of the original document should be clearly mentioned as such.**
7. **In the case of translations, verbatim sentences mentioned in (6.) are preserved in the language of the original document accompanied by verbatim translations to the language of the translated document. All translations state clearly that the author is not responsible for the translated work. This license is included, at least in the language in which it is referenced in the original version.**
8. **Whatever the mode of storage, reproduction or dissemination, anyone able to access a digitized version of this document must be able to make a digitized copy in a format directly usable, and if possible editable, according to accepted, and publicly documented, public standards.**
9. **Redistributing this document to a third party requires simultaneous redistribution of this licence, without modification, and in particular without any further condition or restriction, expressed or implied, related or not to this redistribution. In particular, in case of inclusion in a database or collection, the owner or the manager of the database or the collection renounces any right related to this inclusion and concerning the possible uses of the document after extraction from the database or the collection, whether alone or in relation with other documents.**

Any incompatibility of the above clauses with legal, contractual or judiciary decisions or constraints implies a corresponding limitation of reading, usage, or redistribution rights for this document, verbatim or modified.

End of OR Magazine License

Acknowledgements and contributions

Many people has contributed in one way or another to the writing of this paper. Among them, let us cite Richard Stallman, and several people of the *freesw mail* list[1], which was set up for the discussion of this paper, and is now used also for general discussions about free software issues.

Some contributions from specific people not in the working group are the following:

- Franco Gasperoni provided much of the information regarding ACT and ACT Europe, used in the corresponding business case example.
- Alain Roumiguier provided much of the information regarding Open CASCADE and Matra Datavision, used in the corresponding business case example.

[1] Should you be interested in subscribing to this list, follow directions available at `http://eu.conecta.it`

Contents

1 Introduction

Exactly like the personal computer and the Internet, open source software[2] recently got the attention of the press as a totally new thing that 'suddenly appeared'. Many people believe that the impact of open source software in the information technology industry and in society in general will be huge, and without precedent in its nature, to the point that the current rules by which the software industry behaves will completely change.

The European Commission services, concerned by the effects of this new paradigm on Europe and on the landscape of information technology promoted the creation of a working group on libre software. This document has been written by this group, at the request of the European Commission, with the intent of clarifying the nature of the open source movement, its impact on the way that software products are created, maintained and used, and on the industry that currently produces and maintains such software.

In this document we start by presenting, in section 2, a brief history of open source software, showing that it has a long tradition, and that this heritage is fundamental to the understanding of the peculiar aspects of this not-so-new way of thinking about software. Later on, in section 3, a definition of open source and free software is discussed, providing the reader with a background which will help to understand later sections. In section 4, the development models for open source software are discussed, including the advantages and perceived disadvantages of these models. Some emphasis is also devoted in this section to the issue of how open source models achieve a productive mix of cooperation and competition.

Section 5 introduces some notes on the economy of open source software, including a taxonomy of open source business models (with special emphasis on those that seem to be sustainable), some comments on the impact of the adoption of open source software on total cost of ownership, and some notes on macroeconomic effects. After that, issues related to intellectual property are discussed in section 6, including how copyright and patent law affect open source software, and in section 7 some specific impacts of open source on information technologies (promotion of protocols, diffusion of technology, and security) are presented.

At the end of the document, we describe some possible scenarios for the future of the open source movement (in section 8), and make also some recommendations intended to take better advantage of open source software, and to help to remove the barriers which could hinder its development, for the European Commission and any concerned national government (in section 9). After some conclusions (in section 10), some appendixes with useful information are also included.

2 A brief history of open source software

Although all the stories related to software are obviously short, that of open source software is one of the longest amongst them. In fact, it could be said that in the beginning, there was only free (libre) software. Later on, proprietary[3] software was born, and it quickly dominated the software landscape, to the point that it is today considered as the only possible model by many (knowledgeable) people. Only recently has the software industry considered free software as an option again.

When IBM and others sold the first large-scale commercial computers, in the 1960s, they came with some software which was free (libre), in the sense that it could be freely shared among users, it came with source code, and it could be improved and modified. In the late 1960s, the situation changed after the "unbundling" of IBM software, and in mid-1970s it was usual to find proprietary software, in the sense that users were not allowed to redistribute it, that source code was not available, and that users could not modify the programs.

In late 1970s and early 1980s, two different groups were establishing the roots of the current open source software movement:

- On the US East coast, Richard Stallman, formerly a programmer at the MIT AI Lab, resigned, and launched the GNU Project [26] and the Free Software Foundation. The ultimate goal of the GNU Project was to build a free operating system, and Richard started by coding some programming tools (a compiler, an editor, etc.). As a legal tool, the GNU General Public License (GPL) was designed not only to ensure that the software produced by GNU will remain free, but to promote the production of more and more free software. On the philosophical side, Richard

[2] Throughout this paper, we will use the terms 'open source' and 'free software' to refer to the kind of software under study. As Richard Stallman stresses, the word 'free' in 'free software' is used as in 'free speech', and not as in 'free beer'. In Spanish and French, there is no ambiguity in the use of 'libre' (as opposed to 'gratis'), and therefore this kind of software is some times referred to as 'libre software' (even when speaking in English). The term 'open source software', is being proposed as a synonym for 'free software' and 'libre software' in many environments. It will be the preferred term through this paper, although probably both 'libre software' or 'free software' could be used wherever 'open source' is used. A discussion on the Free Software Foundation view on the use of 'open sourcet' and 'free softwaret' is available in [24].

[3] Through this document, the term proprietary software will be used to denote all non-free software. This is the usual term for that concept in the free software community.

Stallman also wrote the GNU Manifesto, stating that availability of source code and freedom to redistribute and modify software are fundamental rights.

- On the US West coast, the Computer Science Research Group (CSRG) of the University of California at Berkeley was improving the Unix system, and building lots of applications which quickly become "BSD Unix". These efforts were funded mainly by DARPA contracts, and a dense network of Unix hackers around the world helped to debug, maintain and improve the system. During many time that software was not redistributed outside the community of holders of a Unix AT&T licence. But in the late 1980s, it was finally distributed under the "BSD licence", one of the first open source licences. Unfortunately, at that time every user of BSD Unix also needed an AT&T Unix licence, since some parts of the kernel and several important utilities, which were needed for a usable system, were still proprietary.

Other remarkable open source project of that time is TeX (a typesetting system, by Donald Knuth), which formed around it a strong community which still exists today.

During the 1980s and early 1990s, open source software continued its development, initially in several relatively isolated groups. USENET and the Internet helped to coordinate transnational efforts, and to build up strong user communities. Slowly, much of the software already developed was integrated, merging the work of many of these groups. As a result of this integration, complete environments could be built on top of Unix using open source software. In many cases, sysadmins even replaced the standard tools with GNU programs. At that time, many applications were already the best ones in their field (Unix utilities, compilers, etc.). Especially interesting is the case of the X Window System, which was one of the first cases of open source software funded by a consortium of companies.

During 1991-1992, the whole landscape of open source software, and of software development in general, was ready to change. Two very exciting events were taking place, although in different communities:

- In California, Bill Jolitz was implementing the missing portions to complete the Net/2 distribution, until it was ready to run on i386-class machines. Net/2 was the result of the effort of the CSRG to make an unencumbered version of BSD Unix (free of AT&T copyrighted code). Bill called his work 386BSD, and it quickly became appreciated within the BSD and Unix communities. It included not only a kernel, but also many utilities, making a complete operating system. The work was covered by the BSD licence, which also made it a completely free software platform. It also included free software under other licenses (like for instance the GNU compiler).

- In Finland, Linus Torvalds, a student of computer science, unhappy with Tanenbaum's Minix, [4] was implementing the first versions of the Linux kernel. Soon, many people were collaborating to make that kernel more and more usable, and adding many utilities to complete GNU/Linux[5], a real operating system. The Linux kernel, and the GNU applications used on top of it are covered by GPL.

In 1993, both GNU/Linux and 386BSD were reasonably stable platforms. Since then, 386BSD has evolved into a family of BSD based operating systems (NetBSD, FreeBSD, and OpenBSD), while the Linux kernel is healthy evolving and being used in many GNU/Linux distributions (Slackware, Debian, Red Hat, Suse, Mandrake, and many more).

During the 1990s, many open source projects have produced a good quantity of useful (and usually high-quality) software. Some of them (chosen with no special reason in mind) are Apache (widely used as a WWW server), Perl (an interpreted language with lots of libraries), XFree86 (the most widely used X11 implementation for PC-based machines), GNOME and KDE (both providing a consistent set of libraries and applications to present the casual user with an easy to use and friendly desktop environment), Mozilla (the free software project funded by Netscape to build a WWW browser), etc. Of all these projects, GNOME and KDE are especially important, because they address usability by non-technical people. Their results are already visible and of good quality, finally allowing everybody to benefit from open source software. The software being produced by these projects dispels the common myth that open source software is mainly focused on server and developer-oriented systems. In fact, both projects are currently producing lots of desktop personal productivity applications.

The late 1990s are very exciting times with respect to open source software. Open source systems based on GNU/Linux or *BSD are gaining public acceptance, and have become a real alternative to proprietary systems, competing head to head with the market leaders (like Windows NT in servers). In many niches, the best choice is already open source (an outstanding case is Apache as Web server, with a market share consistently over 50%).

The announce of the liberation of Netscape Communicator, in 1998, was the starting point of a rush of many big companies to understand open source software. Apple, Corel and IBM, for instance, are trying different approaches to

[4] Minix is an operating system designed to be useful for learning about the implementation of operating systems, and was at that time partially described in Tanenbaum's classic book "Operating Systems".

[5] Throughout this paper, we will use the term 'GNU/Linux' to refer to the whole operating system, and 'Linux' to refer to the kernel. Although we know that many people use 'Linux' for both concepts, we feel more comfortable acknowledging the job of both the team who built the Linux kernel and the people who contributed to the GNU project. Both made feasible the existence of the whole operating system.

use, promotion or development of open source software. Many companies of all sizes (from the small startup composed of a couple of programmers to the recently public Red Hat) are exploring new economic models to succeed in the competitive software market. The media has also started to give attention to the formerly marginal open source software movement, which is now composed not only of individuals and non-profit organizations, but also of small and medium companies.

Looking at all these events, the following question easily arises: *Are we at the beginning of a new model of the software industry?* This is a question which is difficult to answer, since there is no known and reliable method for looking into the future. However, through this document, we hope to provide readers with some information which can be useful can use to reach their own answer.

3 What is open source software?

It is not easy to define the term 'open source software' with few words, due to the many categories and variants that exist. But it is not too complicated, either, since the idea in itself is simple. Therefore, before using stricter definitions, let us devote a moment to explain, in a relatively informal way, what do we understand as open source software[6].

3.1 General idea of open source software

When we talk, in English, about 'free software', there is a dangerous ambiguity, due to 'free' meaning both 'freedom' and 'gratis'. Therefore, in this document, we will use mainly the term 'open source' when referring to users freedom of use, redistribution, etc., and 'gratis software' when referring to zero acquisition cost. The use of the Spanish and French word 'libre', by the way, has been adopted in many environments to refer to open source software, but will not be used here for the sake of uniformity. Anyway, before going into more detail, it is a good idea to state clearly that open source software does not have to be gratis. Even more, it usually is not, or at least, not completely.

The main features that characterize free (open source) software is the freedom that users have to:

- Use the software as they wish, for whatever they wish, on as many computers as they wish, in any technically appropriate situation.
- Have the software at their disposal to fit it to their needs. Of course, this includes improving it, fixing its bugs, augmenting its functionality, and studying its operation.
- Redistribute the software to other users, who could themselves use it according to their own needs. This redistribution can be done for free, or at a charge, not fixed beforehand[7].

It is important now to make clear that we are talking about freedom, and not obligation. That is, users of an open source program can modify it, if they feels it is appropriate. But in any case, they are not forced to do so. In the same way, they can redistribute it, but in general, they are not forced to do so.

To satisfy those previous conditions, there is a fourth one which is basic, and is necessarily derived from them:

- Users of a piece of software must have access to its source code.

The source code of a program, usually written in a high level programming language, is absolutely necessary to be able to understand its functionality, to modify it and to improve it. If programmers have access to the source code of a program, they can study it, get knowledge of all its details, and work with it as the original author would.

Paradoxically, if this freedom is to be guaranteed for a given piece of software, with current legislation, it is necessary to "protect" it with a licence which impose certain restrictions on the way that it can be used and distributed (as it will be shown later). This fact causes some controversy in certain circles, because it is considered that these licences make the software distributed under them "less free". Another view, more pragmatic, is that software will be made more free by guaranteeing the perpetuation of these freedoms for all its users. Because of that, people holding this view maintain that it is necessary to limit the ways of use and distribution. Depending on the ideas and goals of the authors of a piece of code, they can decide to protect it with several different licences.

[6] Here we are referring to 'open source' software (or 'free software') as a generic term, and not to the Open Source Definition [10], which is of course strict enough, as will be discussed later in this section.

[7] It may be revealing to note that the price charged when redistributing is not related in any way to the price paid when the software was got.

3.2 Open source software licences

With the current legal framework, the licence under which a program is distributed defines exactly the rights which its users have over it. For instance, in most proprietary programs the licence withdraws the rights of copying, modification, lending, renting, use in several machines, etc. In fact, licences usually specify that the proprietor of the program is the company which publishes it, which just sells restricted rights to use it.

In the world of open source software, the licence under which a program is distributed is also of paramount importance. Usually, the conditions specified in licences of open source software are the result of a compromise between several goals which are in some sense contradictory. Among them, the following can be cited (see [2] for a more complete discussion on this topic):

- Guarantee some basic freedoms (redistribution, modification, use) to the users.
- Ensure some conditions imposed by the authors (citation of the author in derived works, for instance).
- Guarantee that derived works are also open source software.

Authors can choose to protect their software with different licences according to the degree with which they want to fulfill these goals, and the details which they want to ensure. In fact, authors can (if they desire) distribute their software with different licences through different channels (and prices)[8] Therefore, the author of a program usually chooses very carefully the licence under which it will be distributed. And users, especially those who redistribute or modify the software, have to carefully study its licence.

Fortunately, although each author could use a different licence for her programs, the fact is that almost all open source software uses one of the common licences (GPL, LGPL, Artistic, BSD-like, MPL, etc.), sometimes with slight variations. To simplify things even more, some organizations have appeared recently which define which characteristics a software licence should have to qualify as an open source software licence. Amongst them, the two most widely known are the Debian Project, which defines the Debian Free Software Guidelines (DFSG, see appendix A.1), and the Open Source Initiative (OSI), which defines "open source" licences[10], and is based on the DFSG. The GNU Project also provides its own definition of free software[4].

It is easy to see from the DFSG that price or availability of source code in itself is not enough to characterize a product as "open source software". The significant point lies in the rights given to the community, to freely modify and redistribute the code or modifications of them, with only the restriction that these rights must be given to all and must be non-revocable.

The differences between open source licences lie usually in the importance that the author gives to the following issues:

- Protection of openness. Some licences insist in that any redistributor maintains the same licence, and hence, recipient's rights are the same, whether the software is received directly from the author, or from any intermediary part.
- Protection of moral rights. In many countries, legislation protects some moral rights, like acknowledgement of authorship. Some licences also provide protection for these matters, making them immune to changes in local legislation.
- Protection of some proprietary rights. In some cases, the "first author" (the party that originally made the piece of software) have some additional rights, which in some sense are a kind of "proprietary" rights.
- Compatibility with proprietary licences. Some licences are designed so that they are completely incompatible with proprietary software. For instance, it can be forbidden to redistribute any software which is a result of a mix of software covered by the licence with any kind of proprietary software.
- Compatibility with other open source licences. Some open source licences are not compatible with each other, because the conditions of one cannot be fulfilled if the conditions imposed by the other are satisfied. In this case, it is usually impossible to mix software covered by those licences in the same piece of software.

Some of the more common open source software licences are as follows:

[8] For instance, there are authors that decide to distribute their software under a licence which guarantees the derived works will be open source software too. However, in some cases, they sell that same software to companies interested in keeping derived works proprietary under a different licence which allows it.

- **BSD (Berkeley Software Distribution)**. The BSD licence covers, among other software, the BSD (Berkeley Software Distribution) releases. It is a good example of a "permissive" licence, which imposes almost no conditions on what a user can do with the software, including charging clients for binary distributions, with no obligation to include source code. In summary, redistributors can do almost anything with the software, including using it for proprietary products. The authors only want their work to be recognized. In some sense, this restriction ensures a certain amount of "free marketing" (in the sense that it does not cost money). It is important to notice that this kind of licence does not include any restriction oriented towards guaranteeing that derived works remain open source. This licence is included verbatim in appendix A.2.

- **GPL (GNU General Public License)**. This is the licence under which the software of the GNU project is distributed. However, today we can find a great deal of software unrelated to the GNU project, but nevertheless distributed under GPL (a notable example is the Linux kernel). The GPL was carefully designed to promote the production of more free software, and because of that it explicitly forbids some actions on the software which could lead to the integration of GPLed software in proprietary programs. The GPL is based on the international legislation on copyright[9], which ensures its enforceability. The main characteristics of the GPL are the following: it allows binary redistribution, but only if source code availability is also guaranteed; it allows source redistribution (and enforces it in case of binary distribution); it allows modification without restrictions (if the derived work is also covered by GPL); and complete integration with other software is only possible if that other software is also covered by GPL. This is not the case with LGPL (GNU Lesser General Public License), also used in the GNU project, which allows for integration with almost any kind of software, including proprietary software. The GPL is included verbatim in appendix A.4. More details about the reasons and implications of the GPL are available in [23].

- **MPL (Mozilla Public License)**. This is the licence made by Netscape to distribute the code of Mozilla, the new version of it network navigator. It is in many respects similar to the GPL, but perhaps more "enterprise oriented".

Other well-known licences are the Qt licence (written by Troll-Tech, the authors of the Qt library), the Artistic licence (one of the licences under which Perl is distributed), and the X Consortium licence (see appendix A.3).

3.3 Case study of a non open source licence: SCSL

Sun Microsystems has released some of its core software technologies under a new licence, the Sun Community Source License (SCSL). This is not an open source licence, but it tries to mimic some of the characteristics of open source licences. In fact, the documents where Sun explains and provides rationale for their licensing schema[10] include the usual arguments in favor of open source licences (although applied to SCSL). However, our opinion is that the points missing (those that would made it really open source) cause, from a practical point of view, most of the advantages of open source software to not be applicable to software covered by SCSL.

Some of the most widely known problems (from an open source point of view) with SCSL are:

- All the code modifications have to be sent to Sun if they are to be distributed. And it is Sun who decides if this modifications are redistributable or not. It is not possible to create even 'experimental' code forks, due to clauses covering compatibility tests which must be passed.

- Sun is in fact using two licences for its code. One of them, suitable for "research and internal deployment", requires no fee. But the other, for "commercial deployment", requires that a fee be paid to Sun. The exact amount of this fee is decided by Sun, who therefore has total control on the code developed by contributors.

- For open source developers, the licence is felt to be dangerous, because it 'taints' them. If they agree to the SCSL and download Sun code, they are prevented from using that or similar technology with an open source licence.

Software developed under SCSL benefits from some characteristics similar to those of open source software (access to source code, some redistribution rights). However, it lacks the synergy found in open source projects[11], due to the limitations of access to source code, integration of fixes and improvements, and the restrictions on redistribution. From a strategic point of view, any company or development group which uses or builds on code covered by SCSL is giving Sun a great control on their development and even marketing plans, which is usually considered undesirable.

[9] This is the case of all software licences, including open source licences. However, the GPL makes an interesting use of this legislation, since it is here used to promote the distribution of software which guarantees far more freedom to the user than usual copyrighted works do. Therefore, some times the software covered by GPL is said to be "copylefted", an interesting word game.

[10] The interested reader could refer to the licence FAQ [16], which includes many links to the licence itself and its rationale. Our discussion in this subsection is mainly based on [17].

[11] See subsection 4.3 for a discussion on the advantages of the open source model.

The open source community is already reacting to this licence with the launch of several projects aimed at recreating SCSL covered technology in a completely 'clean room' environment (without using Sun source code). It is always possible to pay later for accessing the test suites and brand the open source product, or simply ignore that problem, and concentrate on making it as compatible as possible[12]. In general, for a commercial project, it will be much more convenient to use, for instance, GPL instead of SCSL. The resulting software will benefit from all the advantages which it could have with SCSL, but the developer will have full control of the development plan, the distribution policy, and the marketing path.

4 Development models

In addition to the impact that open source software is having on the information technology market, it is also producing many contributions in the field of software development. For instance, according to the classic concepts of software engineering, only a centralized management and a strong control on the access to the source code permits a good, high quality software product. But this assumption has been at least partially defeated by the success of several open source projects, where a large number of developers spread around the world loosely collaborate to build reliable and high quality software products. Rigorous management and a clearly defined design were also considered instrumental in a successful coding project. This view has also been dispelled in practice by the many open source software projects that have succeeded even without a clean initial design and without a formal management process.

4.1 The cathedral and the bazaar

There are several attempts to explain this situation. There are some effects unique to open source projects which seem to overcome the standard assumptions. One of the most notable can be summarized in the quote "given enough eyeballs, all bugs are shallow", attributed to Linus Torvalds. This implies that the debugging phase (where software errors are removed, either through code inspection or software testing) can be parallelized, given some coordination between the software coders. Of course, this can be done much more easily if source code is available (many more eyes can inspect it looking for bugs), and if the software is available to as many programmers as possible.

In any case, nowadays it is clear that open source development models are interesting case studies, where new methods for producing programs are being successfully tested. One of the first studies which highlighted some of the characteristics of these new development models was Eric Raymond's "The Cathedral and the Bazaar" [18]. In it, the author sheds some light on the problem of how successful open software software projects are managed. He analyzes two different categories of software development. One of them, which he calls "cathedral-like development" (as an analogy with how Middle Age cathedrals were built), is characterized by a relatively strong control on design and implementation. The other one, which he calls "bazaar-like development", is based on informal communication between the coders, and several small and medium coding efforts that can be chosen at will by the volunteers (as in a bazaar, where everyone can choose what he wants).

The cathedral-like is the traditional style of software management, although it is also used by some open source projects. In this model, there is a strong control on who can submit patches to the code, and how they are integrated, a clearly defined management structure, and a rigorous plan for code releases. On the other hand, the bazaar-like style uses a loose policy on code releases, and relatively informal control over who can provide patches for bugs or new functionality. The model can be characterized with a sentence: "code often, release often". This usually translates to a situation where long standing stable versions are available for general use, while the project follows on adding new features and fixing bugs in rapidly evolving development versions. While the former are released only from time to time, using higher quality assurance procedures, the latter are released with high frequency, so that programmers and users willing to test and improve those development versions can do it. Every time a new function is added, the code is released as a widely distributed development version. Sometimes this process is even extended to the code management system, so that anyone can get a current "snapshot" of what the programmers are working on at any given time.

Going into further detail, the bazaar-like model works in some sense as an accretion model, where a working application with a reasonable level of usefulness gets more and more functionality and patches, and in this way it gets incrementally better. Due to this incremental development, there is a window of time in the initial phases of development, when the software is still not functional enough to attract attention from other programmers, when some kind of funding would be very convenient (in terms of money, programmer time or other resources) to reach an initial point of usability. After this point is reached, the accretion process can start and show its power.

[12] This is, for example, the philosophy behind the open source library Mesa, a clone of SGI OpenGL, completely compatible with it, but without the legal brand 'OpenGL'. Mesa is now used within commercial and open source software without the official brand tag.

Both cases can be considered as extremes in a continuum of software development models. While open source software projects can use almost any model in that continuum, proprietary software projects can only with great difficulty try models different from the cathedral-like, and thus cannot benefit from the features of the bazaar-like ones.

It is, however, important to understand that open source software can be created with both approaches. In fact, there are cases like the gcc compiler and the XFree86 window system that are well known for following a model quite close to the cathedral-like. The important idea here is that open source software allowed for the first time for the introduction of successful alternative styles of software development. And many of these styles are well suited to several software projects where it is not possible to apply the rigidly defined rules of software production typical of cathedral-style development.

4.2 Developers motivation

Another important point is about motivation. When people approach the open source movement, one of the first questions is something like "why did all those people make such a good software without a clear reward in terms of money?" The answer is difficult to explain only in terms of money and personal expectations. In several cases, there is a clear expectancy of economic reward (this is in fact usually the case when it is a company who leads an open source project). But in many other cases, the reason which impels programmers to start, contribute and maintain open source projects is not directly related to economic rewards. For many people, programming is considered as a highly rewarding activity in itself. As such, contribution to open source projects can even start as a hobby, or as a side effect of some University or School assignment. The reward of coding is also greatly amplified by the fact that the code is in use by people, and that a community starts to gather around and discuss specific functionality, design, and coding issues[13].

This psychological effect is really important in explaining why so many projects are started out of the blue, with seemingly no reward. Although it may seem surprising at first view, it is not that rare if we put it in context. For instance, most of the history of information science and programming, in fact, started this way in academic circles. And still many non-applied sciences advance thanks to the work of scientists who feel more rewarded by research in itself than by money. Although this effect of self-reward is perhaps not so common in the world of proprietary software development, it is today a strong force in the open source community. And what is even more important, it seems clear that it has an extremely good impact on developer's productivity, an interesting effect in a discipline where differences in productivity from person to person are often a matter of orders of magnitude.

However, this not the only force which drives open source software development. Often open source simply makes sense in a practical or economical way, as a more efficient way of producing better software with a given amount of resource. In other cases, a company can enter open source development as the only way to succeed in a market already dominated by strong competitors, or as a way of shortening the time to market of a product (by reusing existing open source code). These more classical motivations, more interested to companies, will be discussed in section 5, where self-sustaining open source based economical models will be presented.

4.3 Advantages of open source software

Motivations for using and developing open source software are mixed, ranging from philosophical and ethical reasons to pure practical issues. In this subsection, some of the most widely proposed practical advantages will be introduced. For a discussion on some of the ethical issues (which is not covered in this document), refer to [25].

Usually, the first perceived advantage of open source models is the fact that open source software is made available gratis or at a low cost. But this characteristic is not exclusive to open source software, and several proprietary software products are made available in similar ways (a prominent case could be Microsoft's Internet Explorer). What really distinguishes open source software from software available without fee is the combination of effects due to the characteristics discussed in section 3.1. All of them combined produce a synergistic impact which is the cause of the real advantages of the open source model. Let us provide some more detail on how do these characteristics turn into advantages:

- *The availability of the source code and the right to modify it* is very important. It enables the unlimited tuning and improvement of a software product. It also makes it possible to port the code to new hardware, to adapt it to changing conditions, and to reach a detailed understanding of how the system works. This is why many experts are reaching the conclusion that to really extend the lifetime of an application, it must be available in source form. In fact, no binary-only application more than 10 years old now survives in unmodified form, while several open source software systems from the 1980s are still in widespread use (although in many cases conveniently adapted

[13] Readers interested in the issue of non-economic rewards of open source developers and their motivations can have a look at [19] and [5]. For an introduction on the issues of motivation and reward in general, the reader can refer to [13].

to new environments). Source code availability also makes it much easier to isolate bugs, and (for a programmer) to fix them.

- *The right to redistribute modifications and improvements to the code*, and to reuse other open source code, permits all the advantages due to the modifiability of the software to be shared by large communities. This is usually the point that differentiates open source software licences from "nearly free" ones. In substance, the fact that redistribution rights cannot be revoked, and that they are universal, is what attracts a substantial crowd of developers to work around open source software projects.

- *The right to use the software in any way*. This, combined with redistribution rights, ensures (if the software is useful enough), a large population of users, which helps in turn to build up a market for support and customization of the software, which can only attract more and more developers to work in the project. This in turn helps to improve the quality of the product, and to improve its functionality. Which, once more, will cause more and more users to give the product a try, and probably to use it regularly.

The issue about non-exclusive rights on the software, which has just being mentioned, deserves some more attention. When no one holds exclusive rights on a given code (sometimes mentioned as "life or death rights"), several traditional problems of the proprietary software model can be overcome:

- *There is no one with the power to restrict in a unilateral way how the software is used*, even in a retroactive way. Such a power manifests, for instance, when a proprietary software vendor decides not to upgrade some software product for some old platform. In this case, customers can only stick to the old version of the software, or switch to another product. If open source software is used, customers can also fund some development for the desired platform, or look for other vendors to provide the upgrades (of the very same product).

- *There is no single entity on which the future of the software depends*. This is a very common concern with proprietary software. Let us say that a company uses a software product, and relies on the software manufacturer for upgrades and continued development. If the software manufacturer closes doors, or decides to discontinue development of the product, no one has the right to take the program and continue development on it, effectively killing its usability in the market. This has happened many times, and this problem is amplified by the recent mergers in the software market, that usually lead to "cannibalization" of some software product to allow just one or two to get to the market. Open source software effectively protects against this, because if the group or company that originated the code decides to stop development, it is always possible to fund another software group to continue the maintenance and improvement, without legal nor practical limitations.

- *No "black boxes" are possible*. This point is so important that open source is now considered by many experts as one of the necessary conditions for dependable applications. There are several reasons for this importance. One of them is related to the dependability of the services provided by a given software. By having the source code available, it is possible to perform a thorough inspection and verify the correctness of the algorithm and the implementation scheme used. This is also possible in part even with closed source or nearly free licences. The difference lies in the fact that users are allowed to modify everything they find appropriate to suit their needs. A glaring example is the Linux kernel and its "international patches", a set of enhancements with legal problems in some countries. These patches include support for encrypted communications, and can be legally used in large parts of the world. The patches have being developed by concerned parties in countries where such a development is allowed, and therefore users in those countries can use those enhancements. With binary only products no inspection is possible, with closed source or nearly free licences inspection is possible, but modifications are forbidden, so the inherent advantage of having source code available is rendered ineffective.

- *There is always the possibility of "forking"*, or creating an alternative code base if the current one is in some way perceived as wrongly managed. This is sometimes considered a disadvantage, having to manage not only one code base, but two. A "fork" is a subdivision of the code base in two different parts, each managed by a different group. Forks happens for technical or licence reasons, for example because a particular release is made under a non-free licence, the previous one is used as a base for subsequent free releases. Technical motivations are common, because there are sometimes many different way to perform a task, and it is not possible to decide which is better. So if the two camps cannot reach a consensus and the user base is large enough the code splits in two, and both continue development. If the reasons for the split are overcome, usually the two camps agree on a reunification. A recent example is the decision to reunify the compilers gcc and egcs, and to make one of them (egcs) the new base compiler. In other cases a fork is used as a way to coordinate work. An example is the Linux kernel, where two distinct code bases are used, one "stable" and one "experimental". This way it is possible to introduce new and potentially dangerous technologies without disrupting the stable ones. So, people interested

in leading edge technologies can try them, and people that uses the Linux kernel in production environments can count on stable and tested features.

But the main point about forking is that it introduces several levels of competition within the model. For instance, before forking, several programmers can work harder to keep everybody happy integrating as many well-engineered features as possible, to prevent a fork by people whose needs are not addressed. After a fork, both branches tend to compete for the user base with very similar products: only good quality and quick improvement can maintain them in the market.

- *No per-copy fees can be asked for modified versions*, and anyone can use the current code base to start new projects. Working knowledge can be gathered at a minimal cost. This is what made Internet software systems such an important factor in the new economy: students, and people trying new technologies were able to integrate and adopt them immediately, without the hurdles of commercial or non-disclosure licence agreements. In addition, the right to freely modify them allowed for the incredible expansion in the number of communication protocols and systems, each perfectly tailored to the needs of their users. This is also a reason for the overwhelming success of the Linux kernel, widely employed by students thanks to its near-zero cost, and subsequently used by the same students in the startups originated by them, when they turn into entrepreneurs after leaving University.

- *There are fewer conflicting priorities due to marketing pressures*. This is a simple consequence of the fact that there is no single commercial entity pushing for precise delivery dates or features that must be supported. Usually open source software is delivered "when it is ready", and when the development team feels that its quality is good enough. This means that software usually does not need as many "service packs", updates and such, reducing the maintenance cost. Of course this could be turned into disadvantage if a product is indefinitely delayed, or if some feature is missing one release after another. But in this case, the competition between projects may help. If a project starts failing to meet the expectations of its users, it often happens that a new project is forked, using the same code base, to fill this gap. This happens especially if a market exists for some new features, or for better quality versions of the application.

- *It provides a new forum for democratic action*. As individuals and companies decide where to make improvements in the system, the collective desires of the community determine the overall direction of progress, and yet without compelling anyone. People with opinions about what direction is best can urge others to join in, request help, and in this way influence the overall direction of progress, but without any elections in which the majority overrule the minority.

4.4 Perceived disadvantages of open source models

Of course, open source development models lead also to the perception of some disadvantages. However, some of them are only disadvantages if we are stick to classical (proprietary) development models, which is of course not the case with open source. Let us review some of them:

- *There is no guarantee that development will happen*. In other words: it is not possible to know if a project will ever reach a usable stage, and even if it reaches it, it may die later if there is not enough interest. Of course, this is also a problem with proprietary software, but it is more evident in the case of open source. Especially when a project is started without strong backing from one or more companies, there is a significant initial gap, when the source base is still immature and the development base is still being built. If it is not possible to get funding or enough programmers cooperating at this stage, the project just "dies", or perhaps slowly fades out. Usually, when it reaches a self-sustaining level, the user and development base is such that it can proceed by itself, without other external incentives. This issue will be revisited later on, when business models are discussed in section 5.

- *There may be significant problems connected to intellectual property*. This point is especially important, now that some countries are accepting software and algorithm patents. It is very difficult to know if some particular method to solve a software problem is patented, and so the community can be considered guilty of intellectual property infringement. Some open source packages are already addressing this issue with switches or patches that enable or disable patented code fragments according to the country where the code is used. In other cases, developers consider source code not as an executable device, but a mere description of how a device (the computer) executes, and therefore uphold the idea that source code is not by itself (in absence of an executable program) covered by patent law even in countries where software patents are accepted. However, this idea has still to be tested in courts, and many people do not believe in its viability. In any case, it still leaves problems for the users, who need the executable programs. Although the issue of software patents is a problem for the whole software industry, open source is probably one of the more clear cases where it can be shown how they harm the regular process of

software development. The specific problems are that availability of source code simplifies the detection of patent infringements by patent holders, and that the absence of a company that holds all the rights on the software also makes it difficult to use the mechanisms in use by companies to defend from patent litigation, like cross-licensing or payment of royalties. These issues will be discussed with some detail in subsection 6.2.

- *It is sometimes difficult to know that a project exist, and its current status*. There is not much advertising for open source software, especially for those projects not directly backed by a company willing to invest resources in marketing campaigns. However, several 'aggregation points' for open source software do exist[14], although in many cases they are usable only by experts, and not by the general public. They are also in many cases very specific to some software category, such as scientific software or database systems. There are only a few 'clearing houses' for open source software and projects, and in many cases they are not really up to date. However, some people see this fact as a market opportunity, and several companies with experience in Internet based information services are approaching open source software with added value services which maintain information useful for people or companies trying to locate or evaluate open source software of some given characteristics. In fact, a new business is emerging for providing consulting and searching services related to the selection of open source applications for specific needs. For the general public, some more education is still needed before the regular user can approach these services and get a solution to her software problems in terms of open source software.

It is extremely important to 'see' through the various interpretations of the advantages and disadvantages of open source, and if possible try to analyze with quantitative methods if open source can be helpful in a given situation, or for a given user or company. From a developer point of view, a good example of the perils of launching an open source project is the Mozilla project (launched by Netscape in 1998), considered by some people a complete failure because it took more than an year to get a first beta out. On the contrary, other people perceive, with respect to the same project, that the open source community was capable of producing a totally new code base (larger than the Linux kernel) starting with only some new infrastructure. In fact, now half of the developers of Mozilla work outside of Netscape, and the Mozilla browser has begun to be embedded in many innovative devices, thanks to its modularity and flexibility (for a detailed discussion on the evolution and results of the Mozilla project, read [6]).

4.5 Cooperation and competition

If the characteristics of open source development models were to be defined by a unique expression, 'cooperation and competition' would probably be the one to choose. Indeed, the combination of both mechanisms is visible in almost any open source project, not to mention when we look at the big picture, where every project and company is in some sense competing with others for resources and 'market acceptance', while collaborating with the reuse of the same code base. Let us provide some detail on how those mechanisms work, and enforce each other in a very productive mixture.

- *Cooperation within open source projects*. Developers participating in the same project, usually cooperate at levels higher than those usual in proprietary software projects. The design phase is usually completely open, with all developers collaborating, and during the coding phase is quite common that a developer reads and fixes bugs in the code being developed by another. The flux of information is usually very high, and problems are solved by consensus among at least the core developers. All of this together, in addition to be more effective, usually causes the developers to be more committed to the project, and makes their work much easier, thanks to the help of other developers.

- *Competition within open source projects*. Within open source projects there is also some degree of competition. Most of them are organized as some kind of meritocracy, where the developer who does more for the project deserves more credit, and is given more weight in decisions. This allows for a certain level of healthy competition between the developers, from which the leaders of the project (when such leaders do exist) usually emerge. This is still a fact not understood by many companies, which need to realize that in order to participate in the decisions about future directions of a project they need to have some respected developers within it.

- *Cooperation between open source projects*. The very nature of open source licences ensure a high degree of collaboration, even between competing companies. This is for instance the case of the many companies selling distributions of GNU/Linux. All of them share the same code base (the GNU/Linux operating system and applications), and whenever one of them fixes a bug, or improves a package, all its competitors have access to the source code, and thus to the same fix or improvement. Therefore, projects and companies in the open source world are compelled to collaborate by the open source model. And when they try to avoid helping their competitors, they have to move away from the open source model, which usually gives them more problems than benefits.

[14] The most widely known is Freshmeat, at `http://freshmeat.net`.

- *Competition between open source projects*. But in a world of limited resources, open source projects also compete between themselves. Projects developing software for the same niche compete with each other in a way similar to companies in that niche competing. In the end, they are forced to maintain high levels of quality, or they will lose users and developers, and will finally fade away. In fact, this is one of the mechanisms which ensures a good quality in open source production. When a project starts failing in terms of perceived quality or management, there is a chance that some of the developers will open an independent development branch. This branch soon becomes an independent project, perhaps using a similar code base, which starts to compete with the old one. This competition usually causes both teams to improve their product, and in the long term, one can become mainstream while the other becomes marginal, or they can diverge giving special attention to different issues. This is, for instance, the case with NetBSD, FreeBSD, and OpenBSD, all of them sharing similar code, each with a different focus, and all of them exhibiting a high quality product.

Usually, this mixture of competition and collaboration is not intentional, but the product of the development model, and of the licenses used in open source projects. In short, everybody is forced to compete by exposing the tools (the source code) they are using, and improvements are quick to spread through the competing projects, in a manner quite different to the traditional software industry. Competition and collaboration are probably the ultimate cause of the high efficiency (in terms of quality and quantity of software produced with a given set of resources) which open source projects reach.

5 Economics of open source

The economic impact of open source models is going to be very high, not only in the software industry, but in society in general. In this section, we discuss in some detail some of the more foreseeable issues related to this impact. Some of them are just projections from current trends, but some others have already arrived. To begin with, several new economic models for open source projects will be presented (externally funded, internally funded, unfunded, and internally used). Later on, the issue of total cost of ownership will be discussed in subsection 5.5. In subsection 5.6 some discussion on the macro-economic consequences of widespread use of open source will be introduced. To finish this section, some impressions on the future of the software market, considering open source models, are discussed in subsection 5.7.

The subsections on new economic models are important because many traditional models of the software industry are heavily based on proprietary software where the income is directly related to per-copy price (particularly in the case of shrink-wrapped software). With some exceptions, these traditional models are not viable with open source software, since income cannot come from selling copies of the software (freedom of redistribution tends to set the price at the point where marginal cost of reproduction is near zero). Therefore, open source business must look for other sources of income.

The taxonomy of models presented in this section is not just an analysis of currently existing models, even considering that some of them have been already tested in the industry. On the contrary, we have tried to create as complete as possible a categorization of models that can be self-sustaining, or at least feasible from a business point of view. However, real examples are added to each categories wherever we know of them. Interested readers can refer to [7] for another taxonomy of open source business models, more focused on what is currently available, and to [8], which includes the results of a survey about how do open source business behave. In addition, in appendix B we provide information about some real business based on free software, which can be used as case examples.

5.1 New economic models: Externally funded ventures

We consider in this category groups or companies which develop open source software through the initiative (at least in the financial sense) of some external organization. Usually those external entities determine how the funds are to be spent, and where the development efforts are headed. The developer entity just follows those more or less strict guidelines. In some sense, it could be said that the external entity ‘sponsors’ the development of some given open source software. In this category, we can distinguish at least three models, according to who funds the project and why. We have called them public funding, ‘needed improvement’ funding, and indirect funding.

5.1.1 Public funding

Working groups or individuals receive funding for the development of a good software product, documentation, test cases or whatever. Usually, the only constraints imposed by the funding entity are that funds must be used to complete the project. This is typical of large computer science projects, and the funding usually comes from universities or from national science grants. In fact, many large projects in radioastronomy, computational chemistry, and biology are funded

this way. In addition, some consortiums for the development of Internet tools and technologies have (or have had) such a funding structure. It is important to notice that in these cases the funding institution is not expecting to recover the investment, or to directly benefit from it. Usually, some expectation of social improvement is the reason for the funding.

5.1.2 'Needed improvement' funding

A company or organization may need a new or improved version of a software package, and fund some consultant or software manufacturer to do the work. Later on, the resulting software is redistributed as open source to take advantage of the large pool of skilled developers who can debug and improve it.

An example of this model is the work commissioned to Macadamian Software by Corel Corporation to improve the Wine Windows emulator. This work allowed Corel to port its entire software suite to several additional platforms (the suite was based in MS-Windows) with a simple recompilation process. This was of course significantly cheaper than accomplishing a complete porting process, or than building a proprietary Windows emulator, since Wine was already partially built. The work (still in progress) significantly improved the quality of the emulator, and the improvements made by Macadamian were integrated directly into the Wine code base.

5.1.3 Indirect funding

A company may decide to fund open source software projects if those projects can create a significant revenue source for related products, not directly connected with source code or software.

An example of this case is O'Reilly & Associates[15], a publisher specialized in technical books. They paid some of the most important Perl programmers as a way to ensure continued development of the language, this way allowing for more sales of Perl-related books (which they publish).

Another example is VA Research, a seller of hardware systems preinstalled with GNU/Linux. They enrolled as developers many important Linux kernel programmers, thus helping to guarantee the continuous development of the Linux kernel. With a better Linux kernel, the areas of possible use of GNU/Linux expand, and with this the sales of Linux hardware. Even when other hardware vendors profit from this growth, VA Linux Systems owns a healthy market share of it, which justifies the spending of money for the general improvement of Linux. For instance, VA Linux is spending about 1 million USD in kernel improvement, while the overall market growth is of about 200 million USD, of which VA Linux systems has about 20% (or 40 millions), which leads to additional revenues of about 4 million USD. This obviously justifies the spending of the 1 million.

Yet another case is the writing of software needed to run hardware, for instance, operating system drivers for specific hardware. In fact, many hardware manufacturers are already distributing gratis software drivers. Some of them are already distributing some of their drivers (specially those for the Linux kernel) as open source software.

On other side, Red Hat[16] is also a good example. They founded the Red Hat Advanced Development Labs, and hired several skilled programmers to work on strategic (for Red Hat) open source software (mainly the GNOME system). By funding this project they helped to create a valuable product for their main product (the Red Hat Linux distribution), while collaborating to develop new open source software.

5.2 New economic models: Internally funded or revenue based

In this category, we include models in which an institution develops or maintains open source software to sell services or products directly related to that software, or to improve their productivity. We have identified the following models of this kind: those based on 'best knowledge here' (with or without constraints), on 'best code here' (also with or without constraints), on 'special' licences, and on 'brand selling'.

5.2.1 'Best knowledge here" without constraints

In this model, a company works as a paid consultant, with contracts granted on the basis of the higher level of knowledge of their employees. Any company can implement this model, as there are no limitations that prevent a competent technician from gaining an arbitrarily deep experience of open source software systems. Of course, this also means that any firm using this model is exposed to the risk of being superseded by someone else, if the level of competence is reached but not maintained. An example of this model is LinuxCare, a firm specializing in consulting on GNU/Linux systems, which also sells support.

[15] `http://www.ora.com`
[16] `http://www.redhat.com`

5.2.2 'Best knowledge here' with constraints

To prevent competitors from 'stealing' customers, a firm can place arbitrary limitations on the process of knowledge sharing, through patents or trade secrets. Of course this is incompatible with the diffusion of software through an open source licence. However, it can be implemented by placing under a more restrictive licence just a small (but fundamental) part of the code, usually considering it as a "black box". This kind of limitation can be introduced just to have an additional legal right, for example to avoid patent litigation. In the past this kind of limit has been avoided by the open source community by recoding the 'black box' and creating an open source alternative, for instance by creating a loadable library with identical interfaces.

5.2.3 'Best code here' without constraints

In this model, a company develops some open source code, and sells consulting and maintenance services on it. This is similar to 'best knowledge here', but with an additional advantage in terms of time, since a competitor needs some months to create a similar code, or to understand all the intricacies of someone else's source. This gives a time advantage to the company or group that creates the software in the first place. Examples are Cygnus solutions and their support of the gcc/egcs compiler and the glibc library. AbiSoft, BitWizard, and Digital Creations are other firms with a similar business model.

5.2.4 'Best code here' with constraints

The preceding model can be completed with some additional limits, usually to get a wider gap of saleability of the code. The most common limit is the 'delayed availability', where the author of the code imposes some artificial limit that decays with time or with successive releases of the software product. An example is Alladin with their GhostScript Postscript-compatible printing engine.

5.2.5 'Special' licences

It is also possible to sell 'exceptions' to an open source licence for specific commercial applications. For example, SleepyCat software distributes the dbm database engine under the GPL, but can grant 'exceptions' under an alternative licence which allows for integration into proprietary projects, which is impossible with the GPLed version (they also provide services as consultants on the same code).

5.2.6 'Brand selling'

If a name gets recognized enough, it becomes a selling media by itself, like regular 'brand selling' in other marketing fields. An example is Red Hat[17], that has promoted its distribution with such success that many people identify GNU/Linux with this specific brand of distribution. This also helps to explain the relatively high market capitalization that Red Hat achieved with its initial public offering in the American stock markets.

5.3 New economic models: Unfunded developments

If there is enough 'network effect', there may be no need for funding, just a minimal effort for the organization of releases and patches. Examples of these kinds of open source projects are the Linux kernel, GNU/Linux distributions like Debian[18], BSD-based operating systems such as FreeBSD[19], NetBSD[20], or OpenBSD[21], and the Mesa OpenGL-like library. These efforts started in many cases as the effort of a single man, or of a small group, and through good organization and volunteer work they created an extended networked structure that maintains the code. Even with some (limited) funding for some projects, all of these efforts become successful without an external grant or without explicit money offerings. In fact, this is the case for hundreds of small open source projects.

5.4 New economic models: Internal use

Some projects can get started as a lower-cost alternative to proprietary systems. In this case, the developer company does not have (at least in the beginning) any plan to get external income related to the sale of the software or services related

[17] http://www.redhat.com
[18] http://www.debian.org
[19] http://www.freebsd.org
[20] http://www.netbsd.org
[21] http://www.openbsd.org

to it. The company develops some system because it is useful for them, and later decides to make it open source, and distribute it widely, just to benefit from the open source development source. Probably they will get some contributions, improvements and bug fixes from external developers interested in the software, and some bug reports. Later on, the product may even reach some market acceptance, and the developer company could even get some economic benefits from it.

For instance, a large enterprise with several thousand desktop computers can decide to create some software internally, and to make this software available under an open source licence to get the benefits of a larger base of developers that may be interested in helping out. The Cisco printing system is a good case example of this kind of model.

5.5 Impact of open source in the total cost of ownership

An analysis of business models is not, per se, enough to justify an interest in open source software in companies not directly related to the development, consulting or maintenance of software. The presence of new opportunities could not be a reason sufficient to extend this model to other more traditional development environments, and especially to companies which main business is not directly related to information technologies (although it could be based on them). For this reason, in this section we are including a simple analysis of the impact of open source on total cost of ownership for a company using information technology. A more detailed analysis can be found in [21].

Total cost of ownership (TCO) is a widely used parameter to measure the effective cost of a software or hardware/software combination within an organization. It tries to capture all the real costs of deploying an information technology system, at all stages. However, it does not try to measure the advantages gained by using a given information technology, for which usually another parameter, the return on investment (ROI) is used.

In this section, we will discuss how the use of open source software compares in terms of TCO with proprietary systems. We do not include any discussion on the impact on the ROI, since it depends on the quality of the information technology system, and its suitability for the goals and the business of the of the organization where it is used. Therefore, assuming similar quality and suitability, whether the system includes open source software or not should not have any impact on the ROI.

Total cost of ownership is the effective, combined cost of acquisition and deployment of an information technology throughout all its perceived useful life. We can subdivide this life in several stages: information gathering, acquisition or creation, deployment, training, common use and maintenance, and abandon or transition to a new technology. Let's discuss all of them:

- *Information gathering and 'draft' project.* This is the 'shopping around' stage mentioned by some magazines. It consists of the acquisition of data on the possible solutions available to the organization. As was already mentioned, it is sometimes difficult (at least for the lesser known projects) to gather information on the existence of potentially useful components and their degree of completeness and usefulness when open source projects are considered. There is not much advertising, no central repository of code and sometimes not even an archive of available information. This makes it sometimes more costly to gather information on open source software.

  However, this should be balanced with the fact that information gathered in this stage is more realistic for open source project than for proprietary systems, since the availability of source code, the simplicity for carrying out tests (usually, just download and run), and the usual direct contact with the user community help to avoid exaggerations or understatements in commercial advertising.

- *Acquisition or creation.* Sometimes the software is just acquired, if a suitable product already exists, but usually some kind of creation (or at least integration) of new software is also needed. Acquisition has, of course, a lower cost with open source software because there are no per-copy or per-use fees. The lower cost is particularly evident if the system must be deployed on many platforms, which is common within large organizations. Creation has also a lower cost, because many more programmers can participate in the making and the debugging of the software, if the organization can work with the framework of an open source project.

- *Deployment.* This is a difficult point to evaluate, because deployment is considered in different ways depending on the author. We will consider the deployment phase as the steps needed to bring the information system to a complete working implementation, and in place on all required platforms. This is different from some common definitions, where deployment is considered to cover only the 'installation' phase, or bringing the software to the computing platform.

  In any case, deployment time is apparently shorter with proprietary software, thanks to better installation strategies. Open source software usually tends to adapt better to many different situations and even operating systems, and this flexibility is paid for in terms of increased complexity. This is rapidly changing, thanks to better installation

and compilation systems[22] which in some cases are even better than proprietary installation procedures. In the long term, as more and more open software projects and companies are giving more attention to the installation phase, there will probably be no noticeable difference. On the other hand, the higher configurability, and the availability of the source code of the installation systems makes further customization possible, and helps in large scale deployment.

- *Training*. This is largely independent of the fact that the system is proprietary or open source. A limited advantage in favor of open source software is given by the possibility to check, by reading the source code, those aspects where documentation is scarce. On the other hand, proprietary software producers tend to create large scale training processes, documentation, and teaching material that is made easily available to all interested parties. In open source software, the first documentation projects started just a few years ago, and in general are still not producing the same kind and quality of documentation.

  However, as more and more companies, including technical publishers, join the open source movement, this is also changing. Once more, in the long term, it is difficult to foresee any difference in documentation or training difficulties due to the software being open source or proprietary. An special case is of course the software developed in-house, where there are obviously no advantages for proprietary software. Perhaps, making the software open source and distributing it to other interested parties will help the development of good quality documentation by third parties, at a reasonable cost.

- *Common use and maintenance*. This is usually the predominant part of the TCO, and this is also the point where the largest advantages of open source software are apparent. The availability of the source code allows for immediate bug fixing and improvement, in contrast to proprietary software, where this is a more difficult and slow process, and sometimes even impossible (for example if the software is no longer supported). Even when it is possible to obtain the source code of a proprietary system, usually bug fixing must be done by the software manufacturer itself, and this renders the code availability useless. On the contrary, in the GNU/Linux world, for example, fixes to critical bugs are usually found within hours of the discovery of the problem, and in general it is always possible to fix anything. Also, enhancements and adaptation to changing computing platforms helps in the achievement of a longer lifetime for the software, and for incremental reuse and improvement.

- *Abandonment or transition to a new technology*. This stage is overlooked in many cases when TCO is calculated, since the abandonment is not really a part of the real use of a technology. However, information technology systems are usually built to replace existing systems, so that the transition process must be streamlined and easy to achieve. Open source software helps in this stage by making it possible to create a temporary code base to ease the transition. It also helps in the data transition itself. This point is of particular importance, because pure data is in many cases the real value of an information system. Data translation is often a difficult process, and can even become impossible to achieve if there is not enough information on the format used for the storage or interchange. However, open source software can be ported and adapted to new architectures, and data can be read and translated to new formats in a clean and portable way. If proprietary systems are involved, data loss or recoding (always an expensive process) may be needed when the only platform that runs the software is no longer supported.

As a conclusion, it can be said that of the many stages that are to be considered for TCO estimation, in some of them there are no significant differences due to the open source nature of the software, and when these differences are against open source, they are hoped to become negligible as the model matures and is used more and more. However, there are at least three stages (acquisition, maintenance and transition) where using open source systems provides clear advantages.

5.6 The big (macro-economic) picture

The existence of free (open source) software has a complex and important impact on the economy as a whole. The fact that information technology becomes more accessible and is more trusted is an enabler for the creation of new markets. The existence of a population of pro-users (users that are also producers of software and information) plays a key role in the take-up of new activities and branches, as illustrated by the example of music, digital photography and personal home pages. New products (for instance appliances), new services (for instance training or information services) and new branches of activity can develop on this basis. In specific fields, such as education, the reduction in total cost of ownership that is enabled by use of open source platforms, and even more the mobilisation of the creative abilities of all players in the field lead to an increased efficiency of the public services. The dynamic stability enabled by open source

[22] Examples of these systems are the autoconf engine from the Free Software Foundation, and the RPM and DEB packaging systems from Red Hat and Debian.

platforms, that is the possibility to have long-term trajectories of innovation, make possible more efficient investment strategies in user industries.

However, while the reality of these macro-economic effects is well documented, their quantification, and their comparison (taking in account possible negative macro-economic impact) is not possible today, because of the lack of adequate detailed statistical data on free and open source usage and its detailed links with economical and non-economical activities. The development of the proper macro-economic indicators is needed if decisions with respect to the impact of open source on the economy of a whole country are to be taken based in facts.

5.7 Open source software and renewed competition in the software market

A common discussion, even within the open source community, is whether open source software models are more or less productive than proprietary models. From our point of view, in most situations, proprietary software can only be profitable if no other competing open source software product exists with similar characteristics. And the open source software model seems to ensure that once it enters a niche with enough energy, it can produce a product capable of competing with any proprietary counterpart. The cases of Gnat and Apache are clear in this respect[23]. And cases like Zope[24] seem to demonstrate that at least some investors have realized that fact.

If this opinion is true, open source software models should all become self-sustaining. That is, as time goes by, enough resources should be collected by open source projects to maintain a steady level of development. And resources are not reduced to money, as the Linux case demonstrates. Important resources are, for instance, developers' time, or beta testers. However, a company producing a new product should consider going open source if there is already a dominant producer in its niche. If not, probably it won't achieve great results, since competing with the proprietary rules is hard for starters, especially in established markets. But going open source, the whole environment, and the rules, change. The dominant producer no longer has all the advantage, and many customers will be motivated enough (by lower costs, by a better product, by more control on the development path, or whatever) to give the open source product a try. And this forces the dominant producer to change its strategy, improving on the quality of the product and service, switching to an open source model or offering any improvement that represents value to the buyer of the software product. Right now, this seems to be affecting even big players, like any Unix vendor which tries to compete with GNU/Linux, or any WWW server vendor which tries to compete with Apache.

6 Intellectual property

As is usually the case with any information technology, the issues related to intellectual property are quite important for open source software. Among the four mechanisms that international legislation provide for protecting intellectual propriety, only three (copyright, patents and trademarks) could be suitable for software. The fourth one, trade secret, is obviously not adequate for open source software, since it requires obscurity (by not having access to source code, in the case of software) or restraint on modifications or resell and redistribution of derived works. None of these requirements are possible with open source software.

In this section we will briefly discuss some of the implications of copyright and patent law when applied to open source software, and the benefits and disadvantages that the open source movement can get from both cases. Trademarks are usually dealt with in similar ways as any other business does, and therefore will not be discussed in detail here. However, they are the basis of many open source business, which deal with the branding of services.

In subsection 3.2, we already discussed several issues relating to open source licences. It was implicit in that discussion that software is covered by copyright law. Unfortunately, in many countries (notably in the United States), copyright is no longer the only way to protect software. Nowadays, patent law is becoming more and more a way of protecting software techniques in some parts of the world.

6.1 Open source and copyright law

Copyright is the most usual method of protection for software products. In fact, open source licences are enforceable because they use, in one form or another, copyright law. The rationale for this use is simple: copyright law, by default, do not allow for redistribution (nor even use) of software. The only way that redistribution can be done is by granting

[23] Gnat is an open source Ada95 compiler based on GCC. The company which maintains and provides support for it (ACT) has shown, for several years, a considerable market share of the Ada compiler market. More information about Gnat can be obtained in `http://www.gnat.com`. Apache is a WWW server which has held a market share of over 50% for several years. It is developed by the Apache group (`http://www.apache.org`), in which are involved (among other developers) people working for companies like IBM and Apple.

[24] Zope (`http://www.zope.org`) is an open source WWW publishing and application server software. The company which is developing Zope (Digital Creations) was advised by some investors to make its product open source software, because of the commercial advantages of that move (including a huge increase in the market value of the company) [3].

specific permission in a licence. And that licence can force the redistributor to fulfill certain conditions. This is how open source licences work. However, as was discussed in section 3.2, they use this mechanism to enforce different conditions, ranging from attribution of authorship (as BSD licence does), to obligation of redistribution of any derived work under the same terms of the original licence (as GPL does).

Most open source licences were designed according to the United States law. Just recently some research has been done about the applicability of some of them in particular countries. These studies are essential for the open source movement, since all the open source models depend, in a large proportion, on the validity of open source licenses, in case they are at some point challenged in courts. For instance, preliminary results seem to conclude that GPL (and many similar licences) are enforceable in at least some European countries, and particularly in France. However, much more work is needed in this direction, to provide open source models with a strong legal basis, especially in countries other than the United States, where most open source licences have been written and studied.

There is also an interesting issue in connection with the copyright of interface specifications, which affects the interoperability of open source programs with proprietary ones. In some cases, several companies have been forced to give access to in-depth information on the workings of a program or operating system, to allow developers to extend and integrate the software component in their systems or programs. This kind of information was usually copyrighted and sold only to registered developers, maintaining a strict control on who and where the information was leaked to the outside. However, this situation collides with several court decisions, which have made the information presented in header files (those containing only the names and calling information of the various routines that compose a software system, that is, the interface of the system) not completely copyrightable. That means that it is legal to rewrite a copyrighted header file, part of an interface, to create a compatible version. The rationale is that the header contains only information about the access points to the routines, and provides no information on the inner workings of the software. This allows open source developers to recreate a compatible version of any library or component for which a header file is available.

6.2 Open source and software patents

Software patents, especially when they are granted for trivial algorithms that can easily be reinvented by many developers, represent a serious threat to individual open source developers and small organizations, who cannot afford the costs of patent litigation. Ironically, the situation is even more crucial for open source software than for black box proprietary software, since the code is directly accessible by the patent holders.

In many cases, companies and individuals are trying to get exclusive right on certain technologies through patents, and recently more and more patents on fundamental algorithms and procedures have been granted, especially in the United States. We believe that this is a potentially dangerous practice, not only for open source software in particular, but for the software industry and software practitioners in general. The relatively long time span of currently issued patents and the breadth of some of them are specially disturbing. Also, there is insufficient control on the existence of previous work, and many patents are issued on obvious and ill-defined concepts. These patents can be used as broad-fire weapons against competitors, especially the most smaller ones, unable to afford the costly legal expenses needed to demonstrate that a patent is invalid.

Several clear examples of silly patents have been already exposed by the specialized press. For instance, one of the latest cases is a patent on a ‘novel’ method to correct Y2K problems, using an obvious and widely known technique[25]. Another example is a recently issued patent on an enhancement to the readability of fonts on liquid crystal display, based on sub-pixel addressing. This technique was not only well known and employed widely on the old Apple II computer, which uses a curiously similar system to create on-screen colors, and that enabled to create a double-hires mode using half pixel shifts and single color addressing, but also presented in several papers through the years.

Open source software is especially vulnerable to patent-based attacks, because only a few open source-based companies have the financial power to protect themselves against patent lawsuits. Also, if a patent is issued on a very broad technology or technique it may be impossible to circumvent the patent and create a patent-free alternative.

For a more detailed discussion on how software patents are avoidable, including many references to relevant papers on software patent matters, the reader can refer to [9].

7 Some specific impacts of open source technology

There are some specific impacts of open source technology which deserve a more detailed discussion, because of their importance. We will devote this section to this discussion. The discussion will consider the promotion of standards, the diffusion of information technology, and the availability of more secure systems.

[25] The patent has been recently reexamined, due to complaints by several other companies.

7.1 Promotion of de-facto and de-jure standards

An important lesson learned in the past is that, at least in the information technology field, imposed standards can have a really hard time in creating a real user base. In this respect, the availability of an open source reference implementation can be instrumental in the diffusion of protocol usage. Popular examples of this fact are many of the protocols and much of the infrastructure that have been instrumental in the deployment of the Internet. The TCP/IP suite was popularized by the open source protocol stack created in Berkeley by the CSRG group (the networking stack in BSD Unix). Sendmail, which still remains the reference implementation for SMTP, ESMTP and advanced email functionality is another example. BIND, also developed at Berkeley, is still the most widely used software implementing the DNS protocol, and the availability of open source versions of X-Windows has been very important in the adoption by many vendors of X11 as a standard protocol for graphical user interfaces.

The importance of an open source reference implementation is twofold:

- Reference code helps in the quick adoption of the protocol by commercial and non-commercial entities.
- It is possible to test compliance against a standard code, to help in the harmonization of the different subtle incompatibilities that can arise when a protocol is under-specified. This was especially instrumental in making TCP/IP the great technological achievement that now powers the Internet.

The reference implementation can also help in the case of ‘aggressive split’ of a protocol. For instance, if a vendor decides to create a slightly incompatible version of the same protocol to gain some advantage against its competitors, the existence of the reference implementation is always a fixing point which can help to avoid that split, if all the other vendors stick with it, which will be usually the case.

Just as it was important in the past to have control over the operating system and the standards of the formats for data interchange, in the future it will be very important to have open data interfaces in networks for electronic commerce and content interchange between different services in the Internet. This is the case, for instance, whenever one server automatically obtains some information from another server. These application protocols should stay open, with open source reference implementations, which any company or individual can use or adapt to suit their needs. Otherwise, anyone wishing to use those services will need to purchase some proprietary software from the company which sets the standard. As these services are more and more part of the basic infrastructure needed by any individual for their everyday life and by any company for its normal functioning, it becomes more and more critical that their normal activities should not be controlled by any given software provider.

7.2 Diffusion of information technology

Open source software is now such a significant amount of code and software systems that it is possible to create completely free systems using only open source software, without commercial components. This can be seen as a significant advantage if it can help reduce the gap between those who ‘have’ and those who ‘have not’ in the field of information technology. In addition, the availability of source code and the freedom of modifications to it allows for specific localization and internationalization projects to succeed. For example it is possible to adapt a software system to foreign character sets and alternative input methods. This flexibility is also important in adapting software to very specialized needs, for example for adaptation to handicapped people.

7.3 Availability of more secure systems

The advent of the Internet and other low cost communication technologies introduces an incredible new opportunity for the creation of a true ‘information society’. However, it also raises several doubts over the security of most information systems. It is widely known that most desktop systems are highly vulnerable to intrusion and other security related problems, and can be considered real hazards if the issue is not adequately controlled.

We can easily see that open source software has a distinct advantage over proprietary systems, since it is possible to easily and quickly identify potential security problems and correct them. Volunteers have created mailing lists and auditing groups to check for security issues in several important networking programs and operating system kernels, and now the security of open source software can be considered equal or better than that of desktop operating systems.

It has also already been shown that the traditional approach of ‘security through obscurity’ leaves too many open holes. Even now that the Internet reaches just a part of the world, viruses and cracker attacks can pose a significant privacy and monetary threat. This threat is one of the causes of the adoption of open source software by many network-oriented software systems.

An additional point is the creation of open source testing packages, which can verify the conformance of the code to some given quality standards. This also helps to overcome the problem of testing software packages that are rapidly

changing, to ensure that the software behaves correctly and according to specifications. Test suites can be created to exercise most of a software package (within the obvious limits of such an approach, not based on rigorous formal verification) and thus guaranteeing the reliability of the code.

7.4 Impact in less developed countries

The impact of open source software in less developed countries can be even larger than in regions with well established information technology sectors. Among the many reasons, the following can be cited:

- **Easy access to software products**. Open source software can be easily available for individuals and companies in less developed countries. It is enough, for instance, that a local organization download the intended software, and burn CDs which can distribute along the whole country at low cost. That organization can be public or private, with money coming from local taxpayers or from international contributions or by expectations of profit. Since there are no per-copy costs (except for distribution costs), the access for low-income population to software can be done in conditions affordable by virtually any country.
- **Cost-effective transfer of software technology**. Developed countries can make cost-effective contributions to less developed countries by helping them to adopt free software technologies. Since there is no royalty or per-copy fees, the cost of this transfer is really low for the contributor country. Contributions could be focused in training, localization, and adaptation to local needs, with a great multiplier factor.
- **Direct access to software technology**. Less developed countries can access to leading edge software technologies without having to pay expensive royalties. Usually, these countries do not have easy access to those technologies, at least during the early stages of their deployment. They only get end-user products, at high prices, and without access to the underlying technology. On the contrary, with free software technologies, they get not only the end-user product, but also access to its source code, including all its implementation details.
- **Possibilities of making developments in advanced technologies**. In the open source world, any developer (be it an individual or a company) in any part of the world has, at least in theory, access to any open source project (provided some minimal infrastructure, like a reasonable Internet access). In fact, it is interesting to note how important free software projects are being leaded from countries with little or no tradition in advanced software development. In the future, we can expect more and more interesting free software developments coming from less developed countries. Since the building blocks available for these developments are accessible to any developer, and since free software distribution channels are almost equally accessible for all projects (for instance, if a given program enters Debian, it is distributed with any Debian-based distribution around the world), these projects have many possibilities of succeeding.

As a conclusion, it can be said that libre software levels the ground, and make it more possible to less developed countries no only to benefit of software technologies, but also to participate in their development. However, the reader should notice that the better general infrastructure available in developed countries (specifically in terms of communications infrastructure, educational system, availability of technical trained people, etc.) gives them a great advantage with respect to free software development and use. However, it is important to notice that even given these advantages, less developed countries still can also get direct advantages of the developments made elsewhere.

8 Some possible scenarios for the future of open source

Forecasting software technology is always risky. But we will try to describe some possible scenarios of how the support of the European Commission and governments could impact on the future of the open source movement. However, the variability of the approaches, and the great mobility of the information technology environment makes it difficult to do ‘crystal ball’ predictions. And, in any case, the open source movement has already proved how it can progress by itself. In fact it is astounding how it has done it so well with so little outside support (including support from governments and big corporations).

That said, let us just sketch the following cases:

- No action is taken by the European Commission and governments.
- Limited support is granted by these institutions to open source software.
- “Aggressive” support to open source software is provided by these institutions.

8.1 No action scenario

Up to the present moment, almost no explicit support has been granted by governments in any part of the world to open source development. Therefore, this scenario assumes that things are going to progress following the same path.

Our consideration in this respect is that currently open source systems are already a noticeable part of our computing environment, especially in the server and development areas, where they are already playing an important role in the information technology infrastructure. As companies are discovering, it is possible to live off open source. And as the Red Hat and VA Research IPOs have showed [15], the investors are already betting with their money on their capability to succeed.

If no action is taken by the European Commission and other governments, we can expect that only private companies and individuals will try to walk this route. Since the movement is already proving to be self-sustaining in both the economic and the technical field, and it is competing head to head with the leaders in several market niches, it seems unreasonable to doubt its future health.

Probably open source software will be used as a standard part of the infrastructure by most governments, without the need of any specific endorsement. Instead of buying some proprietary systems, some government offices will decide to buy open source systems and servers. In fact, this trend is already starting to take shape in several countries (at least in Europe). Open source will be preferred in many situations simply on economic or technical grounds, as a lower cost or technically better alternative to proprietary systems. In this case no specific benefit will be reaped by governments, or by the societies they serve, except for the direct cost savings on licences.

In this scenario (and in part also in the others), some threats for the open source movements will also have to be dealt with. Among them, some which are already being discussed within the open source community are:

- *FUD* (fear, uncertainty, doubt) techniques, used by companies or organizations committed to proprietary software, afraid of losing their market share or supremacy. Until now those techniques have not been a real problem, but the situation could change in the future, as more and more resources are devoted to this effort.

- *Dissolution*, due to systems and licences which can be confused with open source software, causing divisions in the community and in the code base, and in the long term, the loss of some of the advantages of the open source model.

- *Ignorance*, or loss of global vision, by the people composing the open source community. As more and more people come to use and produce open source software, there is the risk that a significant part of them dont't really understand how open source really works, and all the lessons learned by the community over the last few decades. This could lead to the conversion of open source in just another buzzword, without any real meaning for many people, and therefore to the loss of the real advantages of the model.

- *Legal impediments*. The legal framework could make more difficult, or even impossible, the progress of the open source movement. The issue of software patents (discussed in subsection 6.2) is one of the more obvious legal impediments that could be found in the next years

Some of these problems could be alleviated with the help of responsive governments, as will be discussed later.

8.2 Limited support scenario

In a limited support scenario, probably it makes most sense to ask how society, administrations and governments in general, and the European Commission in particular, can benefit from open source software, and how they can use it for their own advantage, rather than the other way around (how can the open source movement benefit from governmental support).

In this case, they will probably invest some time and resources to assess the feasibility of open source software in their areas of interest, and to identify the barriers which could impede their adoption of open source technologies. With time, they will identify some strategic open source projects where they want to contribute, and will understand more clearly some of the benefits in terms of flexibility, usefulness in the entire life cycle, and adaptability. They may be interested in trying open source software for several (especially mission critical) components of their infrastructure.

In the long term, this will give results, especially in the form of greater acceptance of open source software in society in general, because of the amplifier effect that its use in governments has on society. If they finally recognize the benefits of the open source model, probably they will also help to overcome the future problems (specially those related to the legal framework) that have been described.

The recent case of the German government funding the development of GPG (GNU Privacy Guard) because it is found to be of great benefit for society shows how this scenario may develop in the short to medium term.

8.3 Aggressive scenario

We are completely unable to predict what exactly 'aggressive' may mean, because legislation may preclude some specific kinds of action, and because what today may seem 'aggressive', tomorrow can be considered as just common sense and normal business practice.

As an example, we can think of legislative actions on the part of the European Commission and the national governments to give preference to open source solutions whenever they are technically feasible. Another interesting action could be the active promotion (by direct or indirect funding) of the development of open source alternatives to proprietary systems in those areas where it is identified that this is convenient (because of strategic, social or economic reasons). This would create an enormous market for open source consulting and solutions, improve significantly the skills of the European information technology work force, and probably increase the usefulness of information technology systems. Also, these measures should have some measurable impact on the import/export balance for information technology products, currently very biased against Europe because the majority of widely distributed (usually shrinkwrapped) software systems come from the United States.

In fact, the whole matter of the level of support that open source deserves is mainly a matter of betting on its future. If open source software is really going to change the whole landscape of the information technology industry, the support that Europe (or any other country) gives to it can only be transformed into more benefits. If open source is not a passing fashion, but is here to stay, the impact could be similar to that of Internet technologies during the last decade. In this case, the more radically that a society adopts it as a technological enabler, the more benefits that society will get from it. In our opinion, if the open source community becomes strong in any given area of the world, that area has a far greater possibility of competing in a software market with changing rules, and the society in that area can benefit earlier from reduced costs, greater economic activity, and widespread diffusion of new technologies.

9 Recommendations by the working group

Open source software can be considered both a great opportunity and an important resource. Some recommendations regarding open source software have already been made in several political circles in the European Union, including the talk by Erkki Liikanen (Commissioner for Enterprise and Information Society) given at ISSE 99 in Berlin, who said: *"[...] the solution to this problem certainly lies in non proprietary and open source systems. This is the key to unlocking the potential of the desktop computing security market."* In addition, in [11] we can read in the paragraph about system architectures: *"by focusing on open software standards (for example building on Linux) it may be possible to spark European creativity in this area and dramatically reduce our reliance on imports."* Some proposals have also been made to several national governments (French and Italian among others) by both politicians and research entities to study and fund open source software initiatives.

Europe has now the opportunity of participating in, and benefiting from the open source movement. European companies and developers are already a driving force in many open source projects. If open source software is able to change the rules in the information technology industry, the companies and countries which better understand it and are more advanced in its use and knowledge will have a clear competitive advantage. And in any case, in the world of open source software, any party that helps the movement in any part of the world, in addition to have clear benefits, will always help the movement as a whole.

In this context, this working group on libre software formulates the following recommendations, designed to help the community to benefit as much as possible from open source software, and to remove the barriers that could prevent the future development of open source projects. When reading such recommendations, consider that open source software is already behaving rather well from a technical point of view, both in terms of quantity and quality, competing head to head with market leaders in several niches. However, the adoption of free software by companies, organizations and individuals is going to be a slow process. By promoting free software and free software adoption, Europe can accelerate this process, and get as much benefit as possible from this new technology. therefore, consider the recommendations not as "how to help open source software", but "how to help Europe to benefit from open source software".

For better readability, we have organized the recommendations in several categories: technical issues, organization and support, legal issues, and training, promotion and explanation of benefits.

9.1 Technical issues

1. **Promotion of open source reference implementations for any protocol standard**. For new protocols and protocol standards, the creation and maintenance of a reference implementation distributed under an open source licence should be encouraged. This would make it possible to have at least one implementation available for public use,

and to serve as a vendor-neutral reference for any other implementor. In addition, this could mitigate the problem of proprietary implementations adding non-standard features to standard protocols, harming interoperability. These issues are discussed with some detail in subsection 7.1.

2. **Endorsement of neutral data formats and open source tools for managing them**. To facilitate interchange of documents, it should be encouraged the adoption by government and public entities of open and public data formats. The creation and use of open source tools for document parsing and transformation would ensure that the public have available, at a small cost, at least those tools to manipulate and work with such documents.

3. **Promotion of projects to improve quality of free software**. Particular attention should be given to projects that try to maximize the quality of open source software, and its suitability for the user community. In this sense, the creation of regression test suites for fundamental software components, and test systems to evaluate the suitability to specific tasks should be designed and funded. In general, contributing regression test suites to open source systems is a very valuable task, and for testing the quality of software those regression tests are needed anyway.

4. **Promotion of free software in precompetitive research projects financed with public money** When the European Union and member states finance research projects with public money, the delivery of open source products as a result of those projects would ensure that the outcomes are available to the community. It would also help to translate that research into marketable products, be it by the entities participating in the research project, or by third parties.

9.2 Organization and support

1. **Services for organization of information related to open source**. The European Commission would provide resources and mechanisms to collect and organize open source software, and efforts related to its promotion and development. This would be very useful, since currently there is no central point of aggregation for both general interest software, special interest software, and documentation for open source projects. The European Commission could use its connections to the academic and research world, through research projects, to help in building such services.

2. **Funding of open source projects, including the provision of general facilities for open source development**. The current efforts at funding open source projects, already present in the Workprogramme 2000 of the Information Society Technologies programme (IST), could be extended to other programmes. In addition, some form of alternative, low budget scheme should be found to allow for very small projects to get funded quickly without the need of the complete formal approval scheme of the IST submission. Appropriate rules for accountability should be sought to ensure the proper use of public money. A possible implementation could be the creation of schemas similar to those of sourceXchange[26] or cosource[27], where users can propose possible projects. Among other areas, this funding should promote the the development of the missing software elements that can be considered fundamental, facilitate the celebration of developer meetings and in general promote the collaboration between European developers involved in open source software projects. A study on how free software development can be supported, and how funding and support mechanisms interact with free software models can be found in [20]

3. **Promotion of projects related to documentation, translation and localization of free software**. These projects are of paramount importance for the widespread adoption of open source solutions by the public in general, and the small and medium companies in particular.

9.3 Legal issues

1. **Fight software patents at all levels**. The impact of software patents on open source software can be specially harmful. Attention should be paid to ensure that patent legislation cannot be used as a weapon against open source software.

2. **Ensure the freedom to build free software implementations which can interoperate with proprietary interfaces**. To be able of replacing proprietary components, and to interoperate with them, open source software needs the ability of recreating interface (header) files (see subsection 6.1, on copyright and open source software). We believe that this freedom should be maintained and tutelated in any possible way, because header rewriting is

[26] `http://www.sourcexchange.com`
[27] `http://www.cosource.com`

in many cases the only way to circumvent proprietary software libraries. The freedom to use inverse engineering techniques to understand proprietary protocols and interfaces (both hardware and software) is also basic for building free software applications which can interoperate with them.

3. **Improve the legal framework so that calls for tenders are open to free software solutions**. Better rules should be written to allow call for tenders to be open to open source software systems. They should be written with a minimal set of requirements that match the tender needs, instead of amounting to basically seeking the best price for a given proprietary solution.

9.4 Training, promotion and explanation of benefits

1. **Promotion of training and education on free software products**. One of the most important barriers to the adoption of free software is the need of training for users. In many cases, it is difficult to find entities providing training programmes of good quality on free software products. Therefore, any measure in the direction of promoting such programmes would be very interesting. In addition, financial means to help companies to enroll their employees in those programs would be of great benefit to accelerate the adoption of open source technologies, specially in the case of SMEs (small and medium enterprises).

2. **Creation of an office to help institutions take advantage of free software**. This office would help all European countries in the process of evaluation and development of open source software programmes. It can also issue directions and organize informational materials, and advise in the resolution of the few remaining issues, mainly those related to intellectual property. This office could function in a position of parity with respect to other relevant European offices. It could also help by offering advice for the modification of the current policies and criteria for project funding within the IST programmes, to better adapt them to the different needs of open source software.

3. **Specific recommendations of use of free software**. A big effort is needed to help all levels of the administration to consider open source solutions as valid offers. Recommendations on the adoption of open source software at all levels could be used by the European Commission and the national governments. The same could be said in terms of support to small and medium companies willing to benefit from open source. Work Programme 2000 has already included some action lines in this direction. We recommend that the same approaches are taken at other levels and in other programmes.

4. **Research about the economic and social impact of open source software**. The economic and social impact of open source is still difficult to measure and explain. Research in these areas should be supported by the European Commission, as a way of developing tools for better allocating resources for maximizing the benefits of open source in the economic fabric in particular, and in the whole society in general. This should include the development of the proper indicators to enable macro-economic studies of these impacts.

9.5 General remarks

By carrying out all these initiative at the European level instead of only at national or regional levels, we hope that the interested critical mass of human resources related to open source development can be equivalent to that of the United States or any other area of the world. This critical mass should help to convert the good current technical level into products, companies and benefits for the society.

It is important to notice that open source software is easily available, with borders being of little importance, with a really low barrier for adoption, and with its results having a great degree of penetration within user and developer communities all around the world, given the enough quantity and quality of well trained human resources. From this point of view, the European critical mass will be just a part of the worldwide critical mass of people related to this technology, but very important for Europe, if we want to get all the benefits it can provide.

10 Conclusions

We have tried to provide to the reader a relatively detailed and as complete as possible introduction to the open source software landscape. We hope to have shown the main characteristics of this technology, which, although has already a long history, is still unknown to many people. We have also tried to expose the main features of open source software, and the mechanisms which drive the working of open source projects which enable these features. Some notes on the economy of open source have been discussed, and although there is still much to try and discover in this respect (particularly with regard to business models), we hope to have shed some light on the economic feasibility of the model. Some specific impacts on several key aspects of the information technology world have also been discussed.

After this background information, we have ended the document with a non-exhaustive list of recommendations to the European Commission and to the governments of the member states. Our feeling is that open source software has already started to modify the rules in the information technology industry, which will produce enormous changes in in the years to come. Given these facts, it is clear to us that those countries and companies which adopt open source technologies in the short term will have a huge competitive advantage, and that society in general can benefit a lot from this early adoption. Europe is in a good position to take early advantage of open source, and can also help the open source movement to get stronger. This collaboration can only be good for both parties. In the long term, the European culture, technological background, and society organization could match well with the open source model and its unique and productive combination of cooperation and competition.

A Some licences

A.1 The Debian Free Software Guidelines

The next definition of free software is taken from the Debian Policy Manual [12], release 2.4.1.0, date April 14th of 1998. Its authors are Ian Jackson and Christian Schwarz, and is revised by David A. Morris. This definition is included in the Debian distribution of the GNU/Linux system.

The Debian Free Software Guidelines (DFSG) is our definition of 'free' software.

1. **Free Redistribution.** The license of a Debian component may not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license may not require a royalty or other fee for such sale.

2. **Source Code.** The program must include source code, and must allow distribution in source code as well as compiled form.

3. **Derived Works.** The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.

4. **Integrity of The Author's Source Code.** The license may restrict source-code from being distributed in modified form *only* if the license allows the distribution of "patch files" with the source code for the purpose of modifying the program at build time. The license must explicitly permit distribution of software built from modified source code. The license may require derived works to carry a different name or version number from the original software. (This is a compromise. The Debian group encourages all authors to not restrict any files, source or binary, from being modified.)

5. **No Discrimination Against Persons or Groups.** The license must not discriminate against any person or group of persons.

6. **No Discrimination Against Fields of Endeavor.** The license must not restrict anyone from making use of the program in a specific field of endeavor. For example, it may not restrict the program from being used in a business, or from being used for genetic research.

7. **Distribution of License.** The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties.

8. **License Must Not Be Specific to Debian.** The rights attached to the program must not depend on the program's being part of a Debian system. If the program is extracted from Debian and used or distributed without Debian but otherwise within the terms of the program's license, all parties to whom the program is redistributed should have the same rights as those that are granted in conjunction with the Debian system.

9. **License Must Not Contaminate Other Software.** The license must not place restrictions on other software that is distributed along with the licensed software. For example, the license must not insist that all other programs distributed on the same medium must be free software.

10. **Example Licenses.** The "GPL," "BSD," and "Artistic" licenses are examples of licenses that we consider **free**.

A.2 BSD licence

This is the licence applied to the software distributions of the Computer Science Research Group, of the University of California at Berkeley.

This licence was changed in 1999, supressing the clause which forces to include acknowledgements to the copyright holder in advertising materials. See [22] for an explanation on this change.

A.3 X Window System (X Consortium) licence

This is the distribution licence of the X Window System, X11R6 (version 11, release 6).

A.4 GNU General Public License

This is the well known GNU General Public License (GPL), version 2 (June, 1991), which covers the largest part of the software from the Free Software Foundation, and a lot of other programs.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps:

1. copyright the software, and

2. offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

2. You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

3. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

> a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
>
> b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
>
> c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

> a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
>
> b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
>
> c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the program's name and a brief idea of what it does.> Copyright (C) 19yy <name of author>
```

```
This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
```

```
This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
```

```
You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) 19yy name of author Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.  This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.
```

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than 'show w' and 'show c'; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright interest in the program 'Gnomovision' (which makes passes at compilers) written by James Hacker.
```

```
<signature of Ty Coon>, 1 April 1989 Ty Coon, President of Vice
```

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

B Some business case examples

In this section, we provide some case examples of companies with some relationship to open source software. This is in no way an endorsement of the business models of these companies, neither of their products. We just hope that these case examples may help the reader to better understand some of the business models related to open source software being used nowadays in Europe. However, keep in mind that most of the information in this section has been provided by the involved companies, and that they are not necessarily the most related to open source, neither the most successful in this respect.

B.1 Matra Datavision: towards an open source business model

On December 1999, Matra Datavision announced the adoption of an open source strategy for its CASCADE geometric modeling libraries with publication of their source code on the Internet, as the Open CASCADE product, distributed

under a LGPL-like licence. Before that announcement, CASCADE had been marketed using a traditional, proprietary model, with an installed base of about 130 active customers, using 250 development licences and 1,500 run-time licences worldwide.

From a technical point of view, Open CASCADE is a set of components for the development of technical and scientific modeling applications ranging from CAD/CAM/CAE to metrology and measuring machines, biomedical software, 3D geological mapping, optical simulation, product design and styling.

When Matra Datavision decide that Open CASCADE was to be distributed under an open source licence, they gave developers free access to the source code of many 3D geometry data structures (from volume primitives to surface creation), including hundreds of modeling algorithms (such as Boolean operations, hidden line removal, fillets and chamfers and graphic representation of 2D and 3D objects in an Open GL-based viewer). Open CASCADE can input and output using several industry standard formats, thus ensuring exchange of application data with current software environments.

For supporting the product, a new organization, encompassing a 50 member development team, was created to contribute to Open CASCADE and guarantee its smooth operation and utilization.

The importance of this announcement was larger than the importance of the product being made free software, since it was probably the first one in its class made by a large European company (Matra Datavision is a wholly owned subsidiary of Aerospatiale Matra, Europe's second largest aeronautics/defense group). Matra Datavision itself currently supports over 6,000 customers worldwide, with sales and service teams based principally in Europe and North America. Customers are companies of many sizes, primarily in the automotive, aerospace and defense, general manufacturing and machinery, consumer product and electromechanical industries.

B.1.1 Objectives and key factors

According to Matra Datavision, the objectives they were pursuing when they decided to go open source are:

- To create a real differentiating factor through publication of the source code.
- To impose Open CASCADE as a de facto standard, by extending its user-base.
- To focus on standardization and services, and not on software publishing.

To fulfill these objectives, they also identified some (in their opinion) key factors, necessary in order to succeed in the open source market:

- Seduce a community of developers. An international team of interested developers is needed to ensure the development of the product. The 'release often and release early' rule has to be followed, so that external developers are rewarded when their work is included in the official version.
- Master the project management and communication tools. Setup rules (for instance, stating that no code will be included in the official version if not properly documented) have to be specified, and make sure that they are well accepted by the contributing community. It is a good idea to select good tools to manage development, bug reports etc.
- Select the right type of licence. It is important that it is recognized by the community as an open source licence, while in accordance with the strategy of the company. Make sure that applications developed for specific customers are not falling under that licence and can remain proprietary, in case this matters.
- Strongly communicate and capitalize on brand image The web site has to be the obvious reference for anyone interested in participating in the project, buying the product or services, etc.
- Develop an appealing services portfolio, including training packages, support, consulting services, etc.

B.1.2 Benefits for customers and third parties

For customers and other entities in the Open CASCADE market, several benefits are perceived. These benefits have to be communicated to them, so that the real advantages that customers get with free software are perceived. Many of these benefits are generic of almost any free software product.

- Companies using it have more control on the product and development process:

  - They can now be sure that their specific development is based on a perennial, high quality, supported product. Even if Matra decides to discontinue the support of the software, the availability of source code ensures that third parties could provide such support.
  - They can influence the research and development of Open CASCADE, and even participate in it, since large parts of that process are being made with complete openness, and in collaboration with external contributors.
  - They can have an independent specific development strategy in a multi CAD, multi platform environment, using the parts of Open CASCADE that they prefer, but maintaining their own improvements.

- Independent software vendors have a more direct access to the base technology:
  - Entrance fee is low. In fact, it is almost zero, since the software can be studied and tested just by downloading it from the Net.
  - Advantage of the components added by the open source community. All those components are available to the vendor for incorporation into its products.
  - Influence the research and development of the product, by collaborating in the open development model.
  - Enhance their application maintainability, by having knowledge not only of the components they develop, but also from the core components supplied by Matra Datavision, since its source code is available.
  - Increase their notoriety through the component gallery.

- Entities devoted to research and education benefit mainly from the openness of the process:
  - They are able to completely disseminate and share research results, which is usually not possible if they are based on proprietary products.
  - Low adoption fee, since the code (and even binary packages ready to install) are available for free.
  - Availability of source code makes many research projects possible. This is the case, for instance, when modification of the existing code is necessary.

B.1.3 Business model

The proposed business model around Open CASCADE is based on the following sources of income:

- Provide specialized components as proprietary software, with a traditional proprietary model.
- Market proprietary components built by third parties.
- Sell services: training, consulting, hot line, etc.

As the reader many notice, this is a mixed business model, where the investment in free software development and maintenance is recovered in part by selling proprietary software based on it or complementing it.

Within the Open CASCADE development plans, third parties have an outstanding importance as partners. Each partner can play different roles, simultaneously or separately, that have to be defined among the following possibilities:

- Generic software provider, able to download the software and disseminate and support it inside their community while adding value and services.
- Components and facilitators provider, able to bring new components to the existing open source library or to the proprietary component gallery
- Application provider, usually supplier of specific applications and often an independent software vendor.
- Generic software user, able to download the software, and to develop its own specific in-house applications.
- Business integrator, in charge of the support and assistance, and providing services for Open CASCADE project implementation.

All of those partners should help to improve the collection of products and services around Open CASCADE, making it more interesting to potential customers.

Of course, not only advantages were perceived. Some risks where also identified due to the decision to go open source. Some (if not all) of them are common to almost any free software project:

- Kill the product if the project fails. Since it is no longer possible to sell licences in a traditional way, there is some risk of never getting enough resources to continue development.
- Misunderstanding from our service customers, specially due to the ignorance of how the open source model works.
- Total loss of control over the project. Since anybody can release improved versions of Open CASCADE, there is some risk that a third party becomes the preferred source of distribution and services for it. Matra Datavision no longer has a monopoly on the product.
- Main competitors possible reactions. Perhaps they are forced to go open source too, which could dismiss the competitive advantage of Open CASCADE being the only free software product in its market niche.

In the following years it will be seen whether the Matra Datavision decision of making CASCADE an open source product was a right one, and whether it fulfills its expectations.

B.2 ACT and ACT Europe

ACT Europe was founded in 1996 to provide support for commercial, industrial and military uses of the GNAT Professional Ada 95[28] development environment in Europe. It was founded jointly by Ada Core Technologies Inc (ACT), and by the European members of the GNAT Ada 95 project. ACT is a privately held corporation and was founded in August 1994 by the principal authors of the GNAT Ada 95 compiler system. ACT was founded without any initial capital other than some small loans from the principles, and has existed entirely from revenue from its inception. ACT claims to be currently profitable.

The people involved in both ACT and ACT Europe (from now on, ACT) have been working with Ada for over twenty years, starting with the development of a working Ada compiler for preliminary Ada in 1979, and the first validated Ada 83 system in 1983. This work was done at New York University by a team dedicated to the technical success of the Ada language, which moved to ACT after its foundation.

GNAT is the most widely used Ada 95 development system, available on many platforms, from workstations and PCs to bare boards. Ada solutions using GNAT encompass conventional, real-time, embedded, and distributed systems applications. The GNAT technology has always been based on free software, and more specifically on the GNU toolset. The GNAT compiler is integrated with the GCC (the GNU C compiler) back-end. The GNAT debugger is based on GDB (the GNU debugger) that has been adapted for Ada 95. GLIDE, the GNAT integrated development environment, is based on Emacs, which has been adapted and complemented to create a friendlier and complete environment. The GTKAda GUI technology leverages on the GTK graphical toolkit and provides an advanced GUI builder. These are a few but significant examples of the technological offering of ACT. All the products being distributed by ACT are free software, usually under a GPL or LGPL-like licence.

Since its foundation, ACT has established strategic partnerships with hardware and software manufacturers providing Ada 95 capabilities. In the former category is ACT's relationship with Silicon Graphics, Inc., whose new SGI Ada 95 product is based on GNAT. As of 1999, SGI sold over a billion dollars of Ada related software and equipment. ACT has relationships with several other hardware manufacturers such Compaq and Hewlett Packard. Commercial customers of ACT Europe include Aerospatiale, Alenia, BNP, Boeing, British Aerospace, Canal+, CASA, Dasa, Ericsson, Hughes, Lockheed, TRW, etc.

B.2.1 The business model

The business model of ACT is based on the fact that developers need an increasing amount of help and advice to effectively use and fine-tune the tools at their disposal. The amount of information and the complexity of today's technological reality is daunting. In such a context, support is an essential parameter in the quality and time-to-market equation that will rule the software arena for the time to come. Therefore, ACT has made support the heart of its business, helping customers make effective use of the GNAT technology, and trying to reduce the time-to-market of products developed using it. In addition to offering Internet-based and on-site support, ACT also provides consulting, training and customization in all areas of Ada software development.

The major part of ACT's income comes from renewable support contracts, and therefore they are committed to making sure that their customers receive the best possible support. Until now, these contracts have amounted to a steady income stream and to a sustainable business model. ACT is a low overhead company, with 90% of the personnel being highly trained technical people. The marketing and business team is small. This means that they can continue to provide support without needing a large revenue stream.

[28] The Ada 95 programming language is a standardized (ISO/ANSI) object-oriented programming language upwardly compatible with its predecessor Ada 83.

Despite having support as the core of the business model, they have also been able to invest resources in continued development, being the only vendors supporting several features of the language, and a larger set of targets than their competitors. They do all this with an open technology where users have full access to the source code, not only of the runtime, but of the compiler and tools as well.

B.2.2 Advantages for customers

From the point of view of ACT, the free licence policy does benefit supported customers in indirect but important ways. By making the software itself available to all, it leads to a larger user pool, and therefore more and better trained programmers, unrestricted opportunities to test the software before adoption, and an enthusiastic user community whose work contributes to the overall quality for the product. But it does not obviate the need for direct support from ACT.

Because customers are responsible for delivering a reliable and efficient software application in a timely fashion, they cannot live with unpredictable third party black boxes in their system. In addition their job is not to be an expert in the core technology they are using but rather rely on someone else's expertise for that, while they can concentrate on their software mission which ranges from flying a plane, to controlling a set-top box. GNAT, being free software, allows them to build a black-box free system. This greatly reduces risks and improves time-to-market since the customer has permanent, end-to-end control of all the components present in their final application and has direct access, via a support contract, to the the developers of the software they are using.

GNAT is an open technology, which means that anyone could provide support services. The fact that sources are widely available means that there are many people who know the technology well. Many of them are working at ACT, but there are many other very competent people who know GNAT well, some of which work for or with customers. This is one of the great advantages of an open technology. By contrast with other proprietary technologies, where there is no possibility of any expertise developing outside the vendor. Sources can be put into escrow, but very often the sources are not in sufficiently good shape to be usable to anyone.

The GNAT sources are among the best documented and maintained code in the industry. ACT knows that its code is going to be carefully examined by a large community of Ada experts, and therefore they have to be sure that they like what they see, and find it easy to work with GNAT. Several University research groups use GNAT for projects involving modifying and understanding the GNAT code.

C Some dates of the open source software history

This is just a collection of important events in the open source software history. It is not meant to be complete, and for sure many important milestones are missing. But it should provide a view of the whole landscape of the evolution of the open source software movement.

1950's and 1960's: Software is distributed with source code and without restrictions in forums like the IBM SHARE or the DEC DECUS user groups, or the "Algorithms" section of the "Communications of the ACM" journal.

1969 Apr: RFC number 1 describing first software for the Internet (then ARPANET) is published. The free availability of RFCs and specifically of the protocol specifications was a key factor of the the development of Internet.

1972: Source code is freely distributed in academic circles. Example: MIT, circa 1972, HACKMEM (PDP-6/10 assembly language).

1978: Professor Donald E. Knuth from Stanford University begins to work on TeX, a typesetting system distributed as free software.

1983: Richard Stallman writes the GNU Manifesto, in which he calls for a return to the public sharing of software and source code.

1984: GNU Project begins. Developers begin creating a wide range of Unix-like tools, including for instance an editor (Emacs) and a compiler (GCC). The goal is to build a complete free operating system.

1985: MIT based X Consortium distributes the X Window System as free software covered by one of the less restrictive open source licences.

1985: The Free Software Foundation is founded.

1989: Cygnus, the first commercial company devoted to provide commercial support for GNU software and open source software in general, is funded.

1990: The Free Software Foundation announces its intent to build a Unix-like kernel called GNU Hurd. Their goal is to fill in the last major hole in the GNU suite of software for creating a fully open source development system.

1991: William and Lynne Jolitz write a series in Dr. Dobbs Journal on how to port BSD Unix to i386-based PCs. As a result of the open source software developed and collected at the CSRG of University of California at Berkeley, it was possible to have a complete (and free) BSD operating system. This is the starting point for the BSD family of open source operating systems (NetBSD, FreeBSD, OpenBSD).

1991 Aug: Finnish graduate student Linus Torvalds announces that he has been working on an open source Unix-like kernel, using GNU tools such as GCC.

1991 Oct: Linus Torvalds publicly releases the source code for his Unix-like kernel, which is named Linux.

1991 Dec: Linus Torvalds announces the first self-supporting release of Linux, version 0.11. Developers can now work on Linux without using any proprietary tools or operating systems.

1992 Jul: 386BSD 0.1 is released by William and Lynne Jolitz.

1992: The US Air Force awards New York University (NYU) a contract to build an open source compiler for what is now called Ada 95. The NYU team chooses GNU GCC for code generation and calls their compiler GNAT (GNU NYU Ada 95 Translator).

1993 Aug: Ian Murdock creates a new Linux-based distribution called Debian GNU/Linux, developed by a group of volunteers distributed around the world.

1993 Dec: FreeBSD 1.0, one of the first stable descendent of Jolitzes' 386BSD, is available in the Net.

1994: GNAT receives a commercial boost with the incorporation of Ada Core Technologies (ACT) by its original creators. ACT decides to make money by evolving GNAT and selling support services, rather than by selling GNAT itself. Over time and with the help of ACT, GNAT becomes the dominant Ada 95 compiler.

1994 Jan: Debian GNU/Linux (version 0.91), developed by 12 volunteers, is released.

1994: Marc Ewing begins the Red Hat GNU/Linux distribution. Like the Debian distribution, it is intended to improve on the then-dominant Slackware distribution.

1994 Mar: First issue of the Linux Journal is published.

1994 Oct: NetBSD 1.0 released.

1995 Jan: FreeBSD 2.0 is released.

1995 Apr: First official release (0.6.2) of Apache is distributed.

1996: First Conference on Freely Redistributable Software. Cambridge, Massachusetts, USA.

1996 Oct: Announcement of the KDE project, first project for addressing usability problems of open source unices.

1997 Jun: Eric S. Raymond presents his paper "The Cathedral and the Bazaar" on why the Linux software development model works.

1997 Aug: Announcement of GNOME, a KDE 'competitor', born as a reaction to licensing problems related to KDE usage of the Qt library, which was not open source at that time.

1998 Jan: Netscape declares its intent to release the source code for its Navigator browser, in part persuaded by Raymond's paper, .

1998 Feb: Chris Peterson and others coin the term "open source" and register it, to act as a trade mark for free software products.

1998 Apr: Netscape source code is released, and initial fixes and enhancements begin arriving within hours.

1998 Jun: FREENIX, the Freely Redistributable Software Track of the USENIX Technical Conference is devoted to developers and users of open source software.

1998 Jul: Debian 2.0 is released by more that 300 volunteer developers working on more than 1,500 packages.

1998 Jul: KDE 1.0 is released.

1998 Aug 10: Linus Torvalds and Linux appear on the front cover of Forbes Magazine.

1998 Oct: IBM decides to test open source by using Apache on their AS/400 servers.

1998 Oct: Intel and Netscape invest in Red Hat.

1998 Nov: "Halloween" documents (attributed to Microsoft) are leaked to the public by Eric S. Raymond [27]. The documents analyze strengths and weaknesses of open-source software and Linux.

1999 Oct: 'October GNOME', the (for now) most stable release of the GNOME system, is released.

References

[1] David Bollier. The power of openness. why citizens, education, government and business should care about the coming revolution in open source code software, 1999.
Available in `http://www.opencode.org/h2o/`.

[2] Peter L. Deutsch. Licenses for freely redistributable software. In *Proceedings of the First Conference on Freely Redistributable Software*, Cambridge, Massachusetts, USA, February 1996.

[3] Paul Everitt. How we reached the open source business decision.
Available in `http://www.zope.org/Members/paul/BusinessDecision`.

[4] Free Software Foundation. Free software definition. Available in `http://www.gnu.org/philosophy/free-sw.html`.

[5] Free Software Foundation. The GNU manifesto, 1985.
Available in `http://www.gnu.org/philosophy/`.

[6] Frank Hecker. Mozilla at one: A look back and ahead, 1999.
Available in `http://www.mozilla.org/mozilla-at-one.html`.

[7] Frank Hecker. Setting up shop: The business of open-source software, 1999.
Available in `http://people.netscape.com/hecker/setting-up-shop.html`.

[8] Naomi Hoffman. Open source software.
Available in `http://www.kitware.com/vtkhtml/vtkdata/paper1.pdf`.

[9] W. Neville Holmes. The evitability of software patents. *Computer*, 33(3):30–34, March 2000.

[10] The Open Source Initiative. The open source definition, 1998.
Available in `http://www.opensource.org/osd.html`.

[11] Information Society Technologies Advisory Group (ISTAG). Orientations for workprogramme 2000 and beyond, 1999.

[12] Ian Jackson, Christian Schwarz, and David A. Morris. Debian policy manual.
Available in `http://www.debian.org/doc/manuals/debian-policy/`.

[13] Alfie Kohn. Studies find reward often no motivator. *Boston Globe*, 19 January 1987.
Available in `http://www.fsf.org/philosophy/motivation.html`.

[14] Bernard Lang. Free resources and independent technology in the information sectors, 1997.
Available in `http://pauillac.inria.fr/~lang/ecrits/hanoi/` (orginal version, in French), and `http://www.openresources.com/documents/free-resources-independent/` (translation into English).

[15] Linux Magazine. Inside the Red Hat IPO. the story of the first open source public stock offering. *Linux Magazine*, 1999.
Available in `http://www.linux-mag.com/1999-11/redhatipo_01.html`.

[16] Sun Microsystems. Frequently asked questions about the Sun Community Source License.
Available in `http://www.sun.com/software/communitysource/faq.html`.

[17] Richard P.Gabriel and William N.Joy. Sun community source license principles, 1999.
Available in `http://www.sun.com/software/communitysource/`.

[18] Eric S. Raymond. The cathedral and the bazaar, 1998.
Available in `http://tuxedo.org/~esr/writings/cathedral-bazaar/`.

[19] Eric S. Raymond. Homesteading the noosphere, 1998.
Available in `http://tuxedo.org/~esr/writings/homesteading/`.

[20] Thomas Roessler and Kristian Köhntop. How free software development can be supported, 1999.
Available in `http://www.koehntopp.de/kris/artikel/oss_funding/`.

[21] Jean-Paul Smets-Solanes and Benoît Faucon. *Logiciels Libres. Liberté, Egalite, Business*. Edispher, 1999. In French.
Some information on the book (in English) is available in `http://www.freepatents.org/liberty/index_e.html`.

[22] Richard Stallman. The bsd license. Available in `http://www.gnu.org/philosophy/bsd.html`.

[23] Richard Stallman. Copyleft: Pragmatic idealism, 1998.
Available in `http://www.gnu.org/philosophy/pragmatic.html`.

[24] Richard Stallman. Why "free software" is better than "open source", 1998.
Available in `http://www.gnu.org/philosophy/free-software-for-freedom.html`.

[25] Richard Stallman. Why software should not have owners, 1998.
Available in `http://www.gnu.org/philosophy/why-free.html`.

[26] Richard Stallman. The GNU Project. In Chris DiBona, Sam Ockman, and Mark Stone, editors, *Open Sources. Voices from the Open Source Revolution*. O'Reilly & Associates, 1999.
Available in `http://www.ora.com`.

[27] Attributed to Vinod Valloppillil. Halloween-I. open source software, a (new?) development methodology, 1998. Comments by Eric S. Raymond.
Available in `http://www.opensource.org/halloween1.html`.