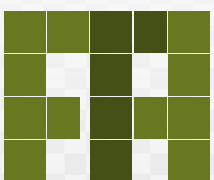# INTRODUCTION TO FREE SOFTWARE

AUTHORS:
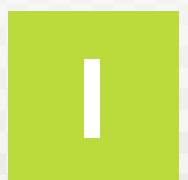J. GONZALEZ BARAHONA
J. SEONE PASCUAL
G. ROBLES

COORDINATORS:
J. MAS HERNANDEZ
D. MEGIAS JIMENEZ

FREE
TECHNOLOGY
ACADEMY

I

Introduction to Free Software

Jordi Mas Hernández (coordinador)
David Megías Jiménez (coordinador)
Jesús M. González-Barahona
Joaquín Seoane Pascual
Gregorio Robles

This version of the text is a preliminary version.
The FTA version will be published soon.

FREE
TECHNOLOGY
ACADEMY

Jordi Mas Hernández

Founding member of Softcatalà and of the telematic network RedBBS. He has worked as a consultant in companies like Menta, Telépolis, Vodafone, Lotus, eresMas, Amena and Terra España.

David Megías Jiménez

Computer Science Engineer by the Universitat Autònoma de Barcelona (UAB, Spain). Master in Advanced Process Automatisation Techniques by the UAB. PhD. in Computer Science by the UAB. Associate Professor in the Computer Science, Multimedia and Telecommunication Department of the Universitat Oberta de Catalunya (UOC, Spain) and Director of the Master Programme in Free Software at the UOC.

Jesús M. González-Barahona

Associate professor in the Department of Telematic Systems and Computing of the Rey Juan Carlos University (Madrid, Spain), where he coordinates the GSyC/LibreSoft research group. His professional areas of interest include the study of free software development and the transfer of knowledge in this field to the industrial sector.

Joaquín Seoane Pascual

Associate professor in the Department of Telematic Systems Engineering of the Technical University of Madrid (Spain), where he has taught courses in programming, protocols, computer architecture, operating systems, Internet services, databases, systems administration and free software. His current interests include the application of ICT in isolated regions of developing countries.

Gregorio Robles

Assistant professor at the Rey Juan Carlos University (Madrid, Spain), where he earned his PhD degree in February 2006. Besides his teaching tasks, he researches free software development from the point of view of software engineering, with special focus in quantitative issues.

This version of the text is a preliminary version. The FTA version will be published soon.

Preface

Software has become a strategic societal resource in the last few decades. The emergence of Free Software, which has entered in major sectors of the ICT market, is drastically changing the economics of software development and usage. Free Software – sometimes also referred to as "Open Source" or "Libre Software" – can be used, studied, copied, modified and distributed freely. It offers the freedom to learn and to teach without engaging in dependencies on any single technology provider. These freedoms are considered a fundamental precondition for sustainable development and an inclusive information society.

Although there is a growing interest in free technologies (Free Software and Open Standards), still a limited number of people have sufficient knowledge and expertise in these fields. The FTA attempts to respond to this demand.

Introduction to the FTA
The Free Technology Academy (FTA) is a joint initiative from several educational institutes in various countries. It aims to contribute to a society that permits all users to study, participate and build upon existing knowledge without restrictions.

What does the FTA offer?
The Academy offers an online master level programme with course modules about Free Technologies. Learners can choose to enrol in an individual course or register for the whole programme. Tuition takes place online in the FTA virtual campus and is performed by teaching staff from the partner universities. Credits obtained in the FTA programme are recognised by these universities.

Who is behind the FTA?
The FTA was initiated in 2008 supported by the Life Long Learning Programme (LLP) of the European Commission, under the coordination of the Free Knowledge Institute and in partnership with three european universities: Open Universiteit Nederland (The Netherlands), Universitat Oberta de Catalunya (Spain) and University of Agder (Norway).

For who is the FTA?
The Free Technology Academy is specially oriented to IT professionals, educators, students and decision makers.

What about the licensing?
All learning materials used in and developed by the FTA are Open Educational Resources, published under copyleft free licenses that allow them to be freely used, modified and redistributed. Similarly, the software used in the FTA virtual campus is Free Software and is built upon an Open Standards framework.

Evolution of this book

The FTA has reused existing course materials from the Universitat Oberta de Catalunya and that had been developed together with LibreSoft staff from the Universidad Rey Juan Carlos. In 2008 this book was translated into English with the help of the SELF (Science, Education and Learning in Freedom) Project, supported by the European Commission's Sixth Framework Programme. In 2009, this material has been improved by the Free Technology Academy. Additionally the FTA has developed a study guide and learning activities which are available for learners enrolled in the FTA Campus.

Participation

Users of FTA learning materials are encouraged to provide feedback and make suggestions for improvement. A specific space for this feedback is set up on the FTA website. These inputs will be taken into account for next versions. Moreover, the FTA welcomes anyone to use and distribute this material as well as to make new versions and translations.

See for specific and updated information about the book, including translations and other formats: *http://ftacademy.org/materials/fsm/1*. For more information and enrolment in the FTA online course programme, please visit the Academy's website: *http://ftacademy.org/*.

I sincerely hope this course book helps you in your personal learning process and helps you to help others in theirs. I look forward to see you in the free knowledge and free technology movements!

Happy learning!

Wouter Tebbens
President of the Free Knowledge Institute
Director of the Free technology Academy

Introduction

"Anyone who hears this, if he can sing, may add and change at pleasure. Let it go from hand to hand: let those who request it have it. As a ball among young women, catch it if you can.

Since this is of 'Good Love', lend it out gladly: do not make a mockery of its name by keeping it in reserve; nor exchange it for money by selling or renting it; for 'Good Love' when bought, loses its charm."

Juan Ruiz, Archpriest of Hita. *The Book of Good Love* (14th century, original in Ancient Spanish)

The first version of these notes was written by Jesús M. González-Barahona, Joaquín Seoane Pascual and Gregorio Robles between April and September 2003. Although we had been discussing for a while preparing a document of this type for the Free Software course that Joaquín and Jesús teach as part of the PhD programs of their respective departments, it was the initiative of the Open University of Catalonia (UOC) that commissioned us to prepare material to introduce the free software masters course, which finally encouraged us to get started. The involvement of Jordi Mas, the academic coordinator of the masters course, in this task was crucial, in that he proposed us for the job and put us in contact with the UOC, additionally supporting our relations with the UOC throughout the project's duration.

Shortly after handing in the first edition, the authors started retouching the material as part of an ongoing process, although with varying degrees of activity, until this second edition was completed in May 2007. During this time, the first edition was extensively used in the free software masters of the UOC and in various other postgraduate courses in Spain and America. The experience with the UOC has been followed with particular interest by Gregorio Robles, who has participated in it, and has therefore obtained feedback that has proven extremely valuable for improving the content. The three of us (Joaquín, Jesús, and since 2006, Gregorio) have also continued with the postgraduate software course at the UPM (Polytechnic University of Madrid) and at the URJC (Rey Juan Carlos University), taking advantage of it in order to test the material.

Once again, the UOC has been the catalyser of this second edition, charging us with a commission that we have taken too long to complete. The work of Jordi Mas and David Megías (of the UOC) has been fundamental, and has provided vital critical support for pushing forward this new edition. The work of José Ignacio Fernández Villamor and Boni García Gutiérrez, pupils of Joaquín Seoane, who have collaborated in reviewing the materials for this second edition, has also been essential.

Previous materials

Some of the texts in these notes are based on previous material, usually belonging to the authors themselves, and in some cases to third parties (used with permission when not completely redrafted). Among them, we would like to mention the following (at the risk of omitting anyone important):

- There are some fragments (especially on the chapters of history and the economy) inspired by the document "Free Software / Open Source: Information Society Opportunities for Europe?" [132], which Jesús González-Barahona co-edited for the European Commission. However, the fragments in question have been extended, retouched and updated to such an extent that in many cases they may be difficult to recognise.

- The section on monopolies and free software (section 5.4) has been based on the paper "Software libre, monopolios y otras yerbas" ("Free software, monopolies and other herbes") [84], by Jesús M. González-Barahona.

- The sections on legislative initiatives and public administration initiatives in relation to free software are partly based on "Iniciativas de las administraciones públicas en relación al Software Libre" ("Initiatives of public administrations related to free software") [103] (thanks to Pedro de las Heras for allowing us to use this material, which he co-authored).

- Part of the section on motives for using free software in the public administrations (Section 6.2) is based on the paper [85], by Jesús M. González-Barahona.

- The chapter on free software engineering is an adaptation of the paper on the state of the art of software engineering applied to free software by Jesús M. González-Barahona and Gregorio Robles for the magazines Novática (Spanish version) and Upgrade (English version).

- In the chapter on case studies, the part regarding the development of Linux is based on a presentation made by Juan-Mariano de Goyeneche during the postgraduate course "Free Programs" of the Polytechnic University of Madrid during academic year 2002-03.

- The historical part of the detailed study of GNOME has been taken from the historical introduction included in the book on "Applications development in GNOME2" prepared by GNOME Hispano and written by one of the authors of this book.

- The FreeBSD case study is partly based on the presentation given by Jesús Rodríguez at the III HispaLinux Conference held in Madrid in the year 2000.

- The Debian and Red Hat case studies are based on the previous work of González-Barahona et al. who have reflected the results of the quantitative analysis of these two distributions in various papers.

- Various materials, especially updates and new material in the chapter on case studies, were prepared by José Ignacio Fernández Villamor and Boni García Gutiérrez towards the beginning of 2007 on a specific branch for modifications made in the context of that year's edition of the postgraduate subject of Joaquín Seoane at the UPM. A large proportion of those materials was included in time for the second edition.

Contents

Glossary

ACM Association for Computing Machinery

AFPL Aladdin Free Public License

ALSA Advanced Linux Sound Architecture

AOL America Online

API Application program interface

ARM Advanced RISC machines

ASCII American standard code for information interchange

AT&T American Telephone & Telegraph

AITC Agency of Information Technologies and Communication

ATK Accessibility Toolkit

BIND Berkeley Internet Name Domain

BIRT Business Intelligence and Reporting Tools

BITNET Because It's There Network

BSA Business Software Alliance

BSD Berkeley Software Distribution

BSDI Berkeley Software Design Incorporated

BSI Bundesamt fur Sicherheit in der Informationstechnik

CDDL Common Development and Distribution License

CD-ROM Compact disc read-only memory

CEPS Cisco Enterprise Print System

CERN Conseil Europeen pour la Recherche Nucléaire

CGI Common Gateway Interface

COCOMO Cost construction model

CORBA Common object request broker architecture

CPL Common Public License

CSRG Computer Systems Research Group

CSS Cascading style sheet

CVS Control version system

DARPA Defense Advanced Research Projects Agency

DBUS Desktop Bus

DCOP Desktop communication protocol

DEC Digital Equipment Corporation

DECUS Digital Equipment Computer User Society

DFSG Debian Free Software Guidelines

DRM Digital rights management

DSDP Device Software Development Platform

DTD Document type definition

DTP Data tools platform

DVD Digital video disk

ECTS European credit transfer scheme

EMP Eclipse Modeling Project

EPL Eclipse Public License

HCEST Higher College of Experimental Sciences and Technology

ETP Eclipse Tools Project

FAQ Frequently asked questions

FDL Free Documentation License

FIC First International Computer

FSF Free Software Foundation

FTP File transfer protocol

FUD Fear, uncertainty, doubt

GCC GNU C Compiler

GDB GNU Debugger

GFDL GNU Free Documentation License

GIMP GNU Image Manipulation Program

GNAT GNU Ada Translator

GNATS GNU Bug Tracking System

GNU GNU's Not Unix

GPL General Public License

GTK GIMP Toolkit

GUADEC GNOME User and Developer European Conference

HIRD HURD of Interfaces Representing Depth

HTML Hypertext markup language

HTTP Hypertext transfer protocol

HURD HIRD of Unix-Replacing Daemons

R&D Research and development

IBM International Business Machines Corporation

IDE Integrated development environment

IEC International Electrotechnical Commission

IETF Internet Engineering Task Force

INRIA Institut National de Recherche en Informatique et en Automatique

IP Internet protocol

IRC Internet Relay Chat

ISO International Standards Organization

ITU International Telecommunications Union

JDK Java Developer Kit

JPEG Joint Photographic Experts Group

JRE Java Runtime Environment

JVM Java Virtual Machine

KBSt Koordinierungs-und Beratungsstelle der Bundesregierung fur Informationstechnik in der Bundesverwaltung

KDE K Desktop Environment

LGPL Lesser General Public License

LISP List processing language

LLC Limited Liability Company

IPA Intellectual Property Act

LTS Long term support

MCC Manchester City Council

MIT Massachusetts Institute of Technology

MPEG Moving Picture Experts Group

MPL Mozilla Public License

MTIC Mission Interministerielle de Soutin Technique pour le Developpement des technologies de l'Information et de la Communication dans l'Administration

NASA National Aeronautics and Space Administration

NCSA National Center for Supercomputing Applications

NPL Netscape Public License

NSFNet National Science Foundation Network

NUMA Non-uniform memory access

NYU New York University

OASIS Organization for the Advancement of Structured Information Standards

ODF Open document format

ODP Open Directory Project

OHGPL OpenIPCore Hardware General Public License

OLPC One Laptop Per Children

WTO World Trade Organisation

WIPO World Intellectual Property Organisation

ORB Object request broker

OSDN Open Software Development Network

OSGi Open Services Gateway Initiative

OSI Open Source Initiative

GDD Gross Domestic Product

PDA Portable digital assistant

PDF Portable document format

PDP Programmed data processor

PHP PHP hypertext preprocessor

PLOS Public Library of Science

PNG Portable network graphics

FAQ Frequently asked questions

QPL Qt Public License

RCP Rich client plaftorm

RDF Resource description framework

RFC Request for comments

RFP Request for proposal

RHAD Red Hat Advanced Development

RPM Red Hat Package Manager

RTF Rich text format

SCO Santa Cruz Operation

SPE Secretariat of Public Education

SGI Silicon Graphics Incorporated

SGML Standard generalised markup language

SISSL Sun Industry Standards Source License

SLS Softlanding Linux System

SOA Service oriented architecture

SPARC Scalable processor architecture

SPICE Simulation program with integrated circuits emphasis

SSL Secure socket layer

TAMU Texas A&M University

TCP Transport control protocol

TEI Text Encoding Initiative

TPTP Test and Performance Tools Project

TRIPS Trade-related intellectual property rights

UMTS Universal mobile telecommunications system

UOC Open University of Catalonia

USA United States of America

USD United States dollar

USENET User network

USENIX Unix Users Group

USL Unix System Laboratories

UUCP UNIX to UNIX copy protocol

VHDL Very high speed integrated circuit hardware description language

W3C World Wide Web Consortium

WIPO World Intellectual Property Organisation

WTO World Trade Organisation

WTP Web Tools Project

WWW World Wide Web

WYSIWYG What you see is what you get

XCF Experimental computing facility format

XML Extensible markup language

Free Software

Jesús M. González-Barahona
Joaquín Seoane Pascual
Gregorio Robles

Index

1. Introduction

"If you have an apple and I have an apple and we exchange apples, then you and I will still each have one apple. But if you have an idea and I have an idea and we exchange these ideas, then each of us will have two ideas."

Attributed to Bernard Shaw

What is free software? What is it and what are the implications of a free program licence? How is free software developed? How are free software projects financed and what are the business models associated to them that we are experiencing? What motivates developers, especially volunteers, to become involved in free software projects? What are these developers like? How are their projects coordinated, and what is the software that they produce like? In short, what is the overall panorama of free software? These are the sort of questions that we will try to answer in this document. Because although free software is increasing its presence in the media and in debates between IT professionals, and although even citizens in general are starting to talk about it, it is still for the most part an unknown quantity. And even those who are familiar with it are often aware of just some of its features, and mostly ignorant about others.

To begin with, in this chapter we will present the specific aspects of free software, focusing mainly on explaining its background for those approaching the subject for the first time, and underlining its importance. As part of this background, we will reflect on the definition of the term (to know what we are talking about) and on the main consequences of using (and the mere existence of) free software.

1.1. The concept of software *freedom*

Since the early seventies we have become used to the fact that anyone commercialising a program can impose (and does impose) the conditions under which the program can be used. Lending to a third party may be prohibited for example. Despite the fact that software is the most flexible and adaptable item of technology that we have, it is possible to impose the prohibition (and it frequently is imposed) to adapt it to particular needs, or to correct its errors, without the explicit agreement of the manufacturer, who normally reserves the exclusive right to these possibilities. But this is just one of the possibilities that current legislation offers: *free software*, on the other hand, offers freedoms that *proprietary software* denies.

> **Proprietary Software**
>
> In this text we will use the term *proprietary software* to refer to any program that cannot be considered free software in accordance with the definition we provide later.

1.1.1. Definition

So, the term *free software*, as conceived by Richard Stallman in his definition (Free Software Foundation, "Free software definition" http://www.gnu.org/philosophy/free-sw.html [120]), refers to the freedoms granted to its receiver, which are namely four:

1) Freedom to run the program in any place, for any purpose and forever.

2) Freedom to study how it works and to adapt it to our needs. This requires access to the source code.

3) Freedom to redistribute copies, so that we can help our friends and neighbours.

4) Freedom to improve the program and to release improvements to the public. This also requires the source code.

The mechanism that guarantees these freedoms, in accordance with current legislation, is distribution under a specific licence as we will see later on (chapter 3). Through the licence, the author gives permission for the receiver of the program to exercise these freedoms, adding also any restrictions that the author may wish to apply (such as to credit the original authors in the case of a redistribution). In order for the licence to be considered free, these restrictions must not counteract the abovementioned freedoms.

The ambiguity of the term *free*

The English term *free software* includes the word *free*, standing for 'freedom', but the term can mean also 'free of charge' or 'gratis', which causes a great deal of confusion. Which is why in some cases the English borrow Spanish/French words and refer to *libre software*, as opposed to *gratis software*.

Therefore, the definitions of free software make no reference to the fact that it may be obtained free of charge: free software and gratis software are two very different things. However, having said this, we should also explain that due to the third freedom, anyone can redistribute a program without asking for a financial reward or permission, which makes it practically impossible to obtain big profits just by distributing free software: anyone who has obtained free software may redistribute it in turn at a lower price, or even for free.

Note

Despite the fact that anyone can commercialise a given program at any price, and that this theoretically means that the redistribution price tends towards the marginal cost of copying the program, there are business models based precisely on selling free software, because there are many circumstances in which the consumer will be prepared to pay in exchange for certain other benefits, such as for example a guarantee, albeit a subjective one, for the software acquired or an added value in the choice, updating and organisation of a set of programs.

From a practical point of view, several texts define more precisely what conditions a licence must fulfil in order to be considered a free software licence. Among these, we would highlight for their historical importance, the free software definition of the Free Software Foundation (http://www.gnu.org/philosophy/free-sw.html) [120], the Debian guidelines for deciding whether a program is free (http://www.debian.org/social_contract.html#guidelines) [104] and the definition of the term *open source* by the Open Source Initiative (http://www.opensource.org/docs/definition_plain.html) [215], which is very similar to the preceding ones.

Note

For example, the Debian guidelines go into the detail of allowing the author to demand that distributed source codes not be modified directly, but rather that the original is accompanied by separate patches and that binary programs be generated with different names to the original. They also demand that the licences do not contaminate other programs distributed by the same means.

1.1.2. Related terms

The term open source software, promoted by Eric Raymond and the Open Source Initiative is equivalent to the term *free software* . Philosophically speaking, the term is very different since it emphasises the availability of the source code and not its freedom, but the definition is practically the same as Debian's ("The open source definition", 1998 http://www.opensource.org/docs/definition_plain.html) [183]. This name is politically more aseptic and emphasises the technical side, which can provide technical benefits, such as improved development and business models, better security, etc. Strongly criticised by Richard Stallman ("Why *free software* is better than *open source*") [204] and the Free Software Foundation (http://www.fsf.org) [27], it has resonated far better with the commercial literature and with the company strategies that one way or another support the model.

Other terms associated in some way to free software are as follows:

Freeware	These are gratis programs. They are normally only distributed in binary format, and can be obtained free of charge. Sometimes it is possible to obtain permission to redistribute, and sometimes not, meaning that then it can only be obtained from the "official" site maintained for that purpose. It is frequently used to promote other programs (normally with more complete functionality) or services. Examples of this type of programs include Skype, Google Earth or Microsoft Messenger.
Shareware	This is not even gratis software, but rather a distribution method since usually the programs can be copied freely, generally without source code, but not used continuously without paying for them. The requirement to pay may be motivated by a limited functionality, being sent annoying messages or the mere appeal to the user's ethic. Also, the licence's legal terms may be used against the transgressor.

Charityware, careware	This is normally *shareware* that requires payment to be directed towards a sponsored *charitable* organisation. In many cases, instead of demanding payment, a voluntary contribution may be requested. Some free software, such as **Vim**, asks for voluntary contributions of this nature (Brian Molenaar, "What is the context of charityware?") [173].
Public domain	Here, the author totally renounces all his rights in favour of the public domain, and this needs to be explicitly stated in the program since otherwise, the program will be deemed proprietary and nothing can be done with it. In this case, if additionally the source code is provided, the program is free.
Copyleft	This is a particular case of free software where the licence requires any distributed modifications to also be free.
Proprietary, locked-in, non-free	These are terms used to refer to software that is neither free nor open source.

1.2. Motivations

As we have seen, there are two large families of motivations for free software development, which likewise give rise to the two names by which it is known:

- The ethical motivation, championed by the Free Software Foundation (http://www.fsf.org) [27], which has inherited the *hacker* culture and supports the use of the term *free*, arguing that software is knowledge that should be shared unimpeded, that hiding it is antisocial and additionally claims that the ability to modify programs is a form of freedom of expression. You can study this in more depth by reading (*Free software, free society. Selected essays of Richard M. Stallman*) [211] or the analysis of Pekka Himanen (*The hacker ethic and the spirit of the information age*. Random House, 2001) [144].

- The pragmatic motivation, championed by the Open Source Initiative (http://www.opensource.org) [54] which supports the use of the term *open source*, and argues the case of the technical and financial advantages that we will discuss in the next section.

Aside from these two main motivations, people working on free software can do so for many other reasons, including for fun (Linus Torvalds and David Diamond, Texere, 2001) [217] or for money, potentially with sustainable *business models*. Chapter 4 studies these motivations in detail on the basis of objective analyses.

1.3. The consequences of the freedom of software

Free software offers many advantages and, of the few disadvantages, many have been exaggerated (or invented) by proprietary competitors. The most well-founded disadvantage is the financial one, since as we have seen it is not possible to make much money from its distribution, which can and tends

to be made by someone other than the author. This is why other business models and financing mechanisms are needed, which we look into in chapter 5. Other disadvantages, such as the lack of support or poor quality, are related to financing but also in many cases are false, since even software with no form of financing tends to offer good support levels thanks to user and developer forums, and often the quality is very high.

Bearing in mind the financial considerations, we should note that the free software cost model is very different to the proprietary software cost model, since a large amount of it develops outside of the formal monetary economy, and frequently using exchange/barter mechanisms: "I give you a program that you are interested in, and you adapt it to your architecture and make the improvements that you need." Chapter 7 discusses the right software engineering mechanisms to make the most of these unpaid for human resources with their own particular features, while chapter 8 studies the tools used to make this collaboration effective. Also, a large share of the costs is reduced by the fact that it is free, since new programs do not need to start from scratch, because they can reuse already made software. The distribution also has a much lower cost, since it is distributed via the Internet and with free advertising through public forums designed for this purpose.

Another outcome of the freedoms is the quality resulting from the voluntary collaboration of people who contribute or discover and notify bugs in environments or situations that are unimaginable for the original developer. Plus, if a program does not offer sufficient quality, the competition may take it and improve on it on the basis of what there is. This is how *collaboration* and *competition*, two powerful mechanisms, combine in order to produce better quality.

Now let's examine the beneficial consequences for the receiver.

1.3.1. For the end user

The end user, whether an individual or a company, can find real competition in a market with a monopoly trend. To be precise, it does not necessarily depend on the software manufacturer's support, since there may be several companies, even small ones with the source code and the knowledge that allows them to do business while keeping certain programs free.

Trying to find out the quality of a product no longer relies so much on the manufacturer's *trustworthiness* as on the guide given by the community's acceptance and the availability of the source code. Also, we can forget the *black boxes*, that must be trusted "because we say so", and the strategies of manufacturers that can unilaterally decide whether to abandon or maintain a particular product.

Evaluating products before they are adopted has been made much easier now, since all we have to do is to install the alternative products in our real environment and test them, whereas for proprietary software we must rely on external reports or negotiate tests with suppliers, which are not always possible.

Because of the freedom to modify the program for own use, users are able to customise it or adapt it to own requirements correcting any errors if there are any. The process of debugging errors found by proprietary software users is normally extremely laborious, if not impossible, since if we manage to get the errors debugged, the correction will often be incorporated in the following version, which may take years to be released, and which moreover we will have to buy again. With free software, on the other hand, we can make corrections or fixes ourselves, if we are qualified, or otherwise outsource the service. We can also, directly or by contracting external services, integrate the program with another one or audit its quality (for example in terms of security). To a great extent, control is passed on from the supplier to the user.

1.3.2. For the public administration

The public administration is a large user of special characteristics, as it has a special obligation towards its citizens, whether to provide accessible services, neutral in relation to manufacturers, or to guarantee the integrity, utility, privacy and security of their data in the long term. All of the above makes it obligatory for the public administration to be more respectful towards standards than private companies and to maintain data in open formats and to process data with software that is independent of usually foreign companies' strategies, certified as secure by an internal audit. Adaptation to standards is a notable feature of free software that proprietary software does not respect to the same extent, because it is generally eager to create captive markets.

Also, the Administration serves as a sort of showcase and guide for industry, meaning that it has a great impact, which ought to be directed at weaving a technological fabric that generates national wealth. This wealth may be created by promoting the development of companies dedicated to developing new free software for the Administration, or maintaining, adapting or auditing existing software. In chapter 6, we will look at this issue in more depth.

1.3.3. For the developer

For the software developer and producer, freedom significantly changes the rules of the game. It makes it easier to continue to compete while being small and to acquire cutting edge technology. It allows us to take advantage of others' work, competing even with another product by modifying its own code, although the copied competitor can then also take advantage of our code (if it is *copyleft*). If the project is well-managed, it is possible to obtain the free collaboration of a large number of people and, also, to obtain access to a vir-

tually free and global distribution system. Nonetheless, the issue of how to obtain financial resources remains, if the software is not the product of a paid-for commission. Chapter 5 deals with this aspect.

1.3.4. For the integrator

For integrators, free software is paradise. It means that there are no longer black boxes that need to be fitted together, often using reverse engineering. Rough edges can be smoothed out and parts of programs can be integrated in order to obtain the required integrated product, because there is a huge shared pool of free software from which the parts can be extracted.

1.3.5. For service and maintenance providers

Having the source code changes everything and puts us in the same position as the producer. If the position is not the same, it is because we are lacking an in-depth knowledge of the program that only the developer has, which means that it is advisable for maintenance providers to participate in the projects that they are required to maintain. The added value of services is much more appreciated because the cost of the program is low. It is currently the clearest business with free software and the one where the most competition is possible.

1.4. Summary

This first chapter has served as a preliminary encounter with the world of free software. The concept defined by Richard Stallman is based on four freedoms (freedom to execute, freedom to study, freedom to redistribute and freedom to improve), two of which require access to the source code. This accessibility and its advantages have motivated another less ethical and more pragmatic point of view, defended by the Open Source Initiative, which has given rise to another term: *open source software* . We have also mentioned other related similar or opposite terms, which serve to clarify various concepts. Finally, we have discussed the consequences of free software for the main parties involved.

2. A bit of history

"When I started working at the MIT Artificial Intelligence Lab in 1971, I became part of a software-sharing community that had existed for many years. Sharing of software was not limited to our particular community; it is as old as computers, just as sharing of recipes is as old as cooking. But we did it more than most. [...] We did not call our software *free software*, because that term did not yet exist; but that is what it was. Whenever people from another university or a company wanted to port and use a program, we gladly let them. If you saw someone using an unfamiliar and interesting program, you could always ask to see the source code, so that you could read it, change it, or cannibalize parts of it to make a new program."

Richard Stallman, "The GNU Project" (originally published in the book *Open sources*) [208]

Although all the histories associated to IT are necessarily brief, free software's is one of the longest. In fact, we could say that in the beginning almost all developed software fulfilled the definition of *free software* , even though the concept didn't even exist yet. Later the situation changed completely, and proprietary software dominated the scene, almost exclusively, for a fairly long time. It was during that period that the foundations were laid for free software as we know it today, and when bit by bit free programs started to appear. Over time, these beginnings grew into a trend that has progressed and matured to the present day, when free software is a possibility worth considering in virtually all spheres.

This history is largely unknown, to such an extent that for many IT professionals proprietary software is software "in its natural state". However, the situation is rather the opposite and the seeds of change that could first be discerned in the first decade of the 21st century had already been sown in the early 1980s.

Bibliography

There are not many detailed histories of free software, and the ones that there are, are usually papers limited to their main subject. In any case, interested readers can extend their knowledge of what we have described in this chapter by reading "Open Source Initiative. History of the OSI" [146] (http://www.opensource.org/docs/history.php), which emphasises the impact of free software on the business community in the years 1998 and 1999; "A brief history of free/open source software movement" [190], by Chris Rasch, which covers the history of free software up until the year 2000, or "The origins and future of open source software" (1999) [177], by Nathan Newman, which focuses to a large extent on the US Government's indirect promotion of free software or similar systems during the decades of the 1970s and the 1980s.

2.1. Free software before free software

Free software as a concept did not appear until the beginning of the 1980s. However, its history can be traced back to several years earlier.

2.1.1. And in the beginning it was free

During the seventies, the IT panorama was dominated by large computers, mainly installed in companies and governmental institutions. IBM was the leading manufacturer, way ahead of its competition. During this period, when buying a computer (the hardware), the software came added. As long as the maintenance contract was paid for, access was given to the manufacturer's software catalogue. Plus, the idea of programs being something "separate" from a commercial point of view was uncommon.

In this period, software was normally distributed together with its source code (in many cases just as source code), and in general, with no practical restrictions. User groups such as SHARE (users of IBM systems) or DECUS (DEC users) participated in these exchanges, and to a certain extent, organised them. The "Algorithms" section of the magazine *Communications of the ACM* was another good example of an exchange forum. We could say that during these early years of IT, software was free, at least in the sense that those who had access to it could normally have access to the source code, and were used to sharing it, modifying it and also sharing these modifications.

On 30[th]June 1969, IBM announced that as of 1970, it would sell part of its software separately (Burton Grad, 2002) [131]. This meant that its clients could no longer obtain the programs they needed included in the price of the hardware. Software started to be perceived as something with an intrinsic value, and consequently, it became more and more common to scrupulously restrict access to the programs and the possibility of users sharing, modifying or studying the software was limited as much as possible (technically and legally). In other words, the situation changed to the one that continues to be case in the world of software at the beginning of the 21st century.

Bibliography

Readers interested in learning about this transition period, can read, for example "How the ICP Directory began" [226] (1998), in which Larry Welke discusses how one of the first software catalogues not associated to a manufacturer was born, and how during this process it was discovered that companies would be prepared to pay for programs not made by their computer manufacturers.

In the mid-1970s it was already totally common, in the field of IT, to find proprietary software. This meant an enormous cultural change among professionals who worked with software and was the beginning of a flourishing of a large number of companies dedicated to this new business. It would still be almost a decade before what we now know as *free software* started to appear in an organised manner and as a reaction to this situation.

2.1.2. The 70s and early 80s

Even when the overwhelming trend was to explore the proprietary software model, there were initiatives that showed some of the characteristics of what would later be considered free software. In fact, some of them produced free software as we would define it today. Of these, we would mention SPICE, TeX and Unix, which is a much more complex case.

SPICE (Simulation Program with Integrated Circuit Emphasis) is a program developed by the University of California, in Berkeley, in order to simulate the electrical characteristics of an integrated circuit. It was developed and placed in the public domain by its author, Donald O. Pederson, en 1973. SPICE was originally a teaching tool, and as such rapidly spread to universities worldwide. There it was used by many students of what was then an emerging discipline: integrated circuits design. Because it was in the public domain, SPICE could be redistributed, modified, studied. It could even be adapted to specific requirements, and that version could be sold as a proprietary product (which is what a large number of companies have done dozens of times throughout their history). With these characteristics, SPICE had all the cards to become the industry standard, with its different versions. And indeed, that is what happened. This was probably the first program with free software characteristics that for a certain period captured a market, the one of integrated circuits simulators, and that undoubtedly was able to do so precisely thanks to these characteristics (in addition to its undeniable technical qualities).

Bibliography

More information on the history of SPICE can be consulted in "The life of SPICE", presented during the Bipolar Circuits and Technology Meeting, Minneapolis, MN, USA, in September 1996 [175].

You can find the SPICE web page at http://bwrc.eecs.berkeley.edu/Classes/IcBook/SPICE/.

Donald Knuth started to develop TeX during a sabbatical year, in 1978. TeX is an electronic typography system commonly used for producing high-quality documents. From the start, Knuth used a licence that today would be considered a free software licence. When the system was considered sufficiently stable, in 1985, he maintained that licence. At that time, TeX was on the largest and most well-known systems that could be considered free software.

Bibliography

You can find some of the milestones in the history of TeX by consulting online http://www.math.utah.edu/software/plot79/tex/history.html [39]. For further details, the corresponding article in Wikipedia is also extremely useful, http://www.wikipedia.org/wiki/TeX [233].

2.1.3. The early development of Unix

Unix, one of the first portable operating systems, was originally created by Thompson and Ritchie (among others) from AT&T's Bell Labs. It has continued to develop since its birth around 1972, giving rise to endless variants sold (literally) by tens of companies.

In the years 1973 and 1974, Unix arrived at many universities and research centres worldwide, with a licence that permitted its use for academic purposes. Although there were certain restrictions that prevented its free distribution, among the organisations that did possess a licence the functioning was very similar to what would later be seen in many free software communities. Those who had access to the Unix source code were dealing with a system that they could study, improve on and extend. A community of developers emerged around it, which soon gravitated towards the CSRG of the University of California, in Berkeley. This community developed its own culture, which as we will see later, was very important in the history of free software. Unix was, to a certain extent, an early trial for what we would see with GNU and Linux several years later. It was confined to a much smaller community, and the AT&T licence was necessary, but in all other aspects, its development was very similar (in a far less communicated world).

Development methods inherent to free software

In *Netizens. On the history and impact of Usenet and the Internet* (IEEE Computer Society Press, 1997 [139], page 139) we can read a few lines that could refer to many free software projects: "Contributing to the value of Unix during its early development, was the fact that the source code was open and available. It could be examined, improved and customised".

Page 142 of the same work states the following: "Pioneers like Henry Spencer agree on how important it was to those in the Unix community to have the source code. He notes how having the sources made it possible to identify and fix the bugs that they discovered. [...] Even in the late 1970s and early 1980s, practically every Unix site had complete sources".

The text of Marc Rochkind "Interview with Dick Haight" is even more explicit (*Unix Review*, May 1986) [198]: "that was one of the great things about Unix in the early days: people actually shared each other's stuff. [...] Not only did we learn a lot in the old days from sharing material, but we also never had to worry about how things really worked because we always could go read the source."

Over time, Unix also became an early example of the problems that could arise from proprietary systems that at first sight "had some free software feature". Towards the end of the 1970s and especially during the decade of the 1980s, AT&T changed its policy and access to new versions of Unix became difficult and expensive. The philosophy of the early years that had made Unix so popular among developers, changed radically to such an extent that in 1991 AT&T even tried to sue the University of Berkeley for publishing the Unix BSD code that Berkeley's CSRG had created. But this is another story that we will pick up on later.

2.2. The beginning: BSD, GNU

All of the cases discussed in the previous section were either individual initiatives or did not strictly comply with the requirements of free software. It was not until the beginning of the 1980s that the first organised and conscious projects to create systems comprising free software appeared. During that period, the ethical, legal and even financial grounds of these projects started to be established (probably more importantly), with them being developed and completed right up to the present day. And since the new phenomenon needed a name, this was when the term *free software* was first minted.

2.2.1. Richard Stallman, GNU, FSF: the free software movement is born

At the beginning of 1984, Richard Stallman, who at the time was employed by the MIT AI Lab, quited his job to started working on the GNU project. Stallman considered himself to be a *hacker* of the kind that enjoys sharing his technological interests and his code. He didn't like the way that his refusal to sign exclusivity or non-sharing agreements made him an outcast in his own world, and how the use of proprietary software in his environment left him impotent in the face of situations that could easily be resolved before.

His idea when he left the MIT was to build a complete software system, for general use, but totally free ("The GNU Project", DiBona *et al.*) [208]. The system (and the project that would be responsible for making it come true) was called GNU ("GNU's not Unix", recursive acronym). Although from the beginning the GNU project included software in its system that was already available (like TeX or, later, the X Window system), there was still a lot to be built. Richard Stallman started by writing a C compiler (GCC) and an editor (Emacs), both of which are still in use today (and very popular).

From the start of the GNU project, Richard Stallman was concerned about the freedoms that the users of the software would have. He wanted not only those who received programs directly from the GNU project to continue to enjoy the same rights (modification, redistribution, etc.) but also those who received it after any number of redistributions and (potentially) modifications. For this reason he drafted the GPL licence, probably the first software licence designed specifically in order to guarantee that a program would be free in this way. Richard Stallman called the generic mechanism that these GPL type licences use in order to achieve these guarantees, *copyleft*, which continues to be the name of a large family of free software licences (Free Software Foundation, GNU General Public Licence, version 2, June 1991) [118].

Richard Stallman also founded the Free Software Foundation (FSF) in order to obtain funds, which he uses to develop and protect free software, and established his ethical principles with the "The GNU Manifesto" (Free Software Foundation, 1985) [117] and "Why software should not have owners" (Richard Stallman, 1998) [207].

From a technical point of view, the GNU project was conceived as a highly structured endeavor with very clear goals. The usual methodology was based on relatively small groups of people (usually volunteers) developing one of the tools that would then fit perfectly into the complete jigsaw (the GNU system). The modularity of Unix, on which this project was inspired, fully coincided with that idea. The method of working generally implied the use of Internet, but because at that time it was not extensively implanted, the Free Software Foundation would also sell tapes on which it would record the applications, which means that it was probably one of the first organisations to obtain financial compensation (albeit in a rather limited way) from creating free software.

In the early 90s, about six years after the project was founded, GNU was very close to having a complete system similar to Unix. However, at that point it had not yet produced one of the key parts: the system's core (also known as the *kernel*, the part of the operating system that relates with the hardware, abstracts it, and allows applications to share resources, and essentially, to work). However, GNU software was very popular among the users of several different variants of Unix, at the time the most commonly used operating system in businesses. Additionally, the GNU project had managed to become relatively well known among IT professionals, and especially among those working at universities. In that period, its products already had a well-deserved reputation for stability and good quality.

2.2.2. Berkeley's CSRG

Since 1973, the CSRG (Computer Science Research Group) of the University of California at Berkeley had been one of the centres where most of the Unix-related developments had been made, especially between 1979 and 1980. Not only were applications ported and other new ones built to function on Unix, but also important improvements were made to the kernel and a lot of functionality had been added. For example, during the 80s, several DARPA contracts (under the US Ministry of Defence) financed the implementation of TCP/IP which until today has been considered the reference for the protocols that make the Internet work (in the process, linking the development of the Internet and the expansion of Unix workstations). Many companies used the CSRG's developments as the bases for their Unix versions giving rise to well-known systems at the time, such as SunOS (Sun Microsystems) or Ultrix (Digital Equipment). This is how Berkeley became one of the two fundamental sources of Unix, together with the "official", AT&T.

In order to use all of the code that the CSRG produced (and the code of the collaborators of the Unix community which to some extent they coordinated), it was necessary to have AT&T's Unix licence, which was becoming increasingly difficult (and expensive) to obtain, especially if access to the system's source code was required. Partly in an attempt to overcome this problem, in June 1989 the CSRG released the part of Unix associated to TCP/IP (the implementation of the protocols in the kernel and the utilities), which did not include AT&T code. It was called the *Networking Release 1* (Net-1). The licence with which it was released was the famous *BSD licence*, which except for certain problems with its clauses on advertising obligations, has always been considered an example of a minimalist free software licence (which in addition to allowing free redistribution, also allows incorporation into proprietary products). In addition, the CSRG tested a novel financing model (which the FSF was already trying out successfully): it sold tapes with its distribution for USD 1,000 each. Despite the fact that anybody in turn could redistribute the content of the tapes without any problem (because the licence allowed it), the CSRG sold tapes to thousands of organisations thus obtaining funds with which to continue developing.

Having witnessed the success of the Net-1 distribution, Keith Bostic proposed to rewrite all of the code that still remained from the original AT&T Unix. Despite the scepticism of some members of the CSRG, he made a public announcement asking for help to accomplish this task, and little by little the utilities (rewritten on the basis of specifications) started coming in to Berkeley. Meanwhile, the same process was done with the kernel, in such a way that most of the code that had not been produced by Berkeley or volunteer collaborators was rewritten independently. In June 1991, after obtaining permission from the University of Berkeley's governing body *Networking Release 2* (Net-2) was distributed, with almost all of the kernel's code and all of the utilities of a complete Unix system. The set was once again distributed under the BSD licence and thousands of tapes were sold at a cost of USD 1,000 per unit.

Just six months after the release of Net-2, Bill Jolitz wrote the code that was missing for the kernel to function on the i386 architecture, releasing 386BSD, which was distributed over the Internet. On the basis of that code later emerged, in succession, all the systems of the *BSD family: first NetBSD appeared, as a compilation of the patches that had been contributed over the Net in order to improve 386BSD; later FreeBSD appeared, as an attempt to focus on the support of the i386 architecture; several years later the OpenBSD project was formed, with an emphasis on security. And there was also a proprietary version based on Net-2 (although it was certainly original, since it offered its clients all the source code as part of the basic distribution), which was done independently by the now extinct company BSDI (Berkeley Software Design Inc.).

Partly as a reaction to the distribution produced by BSDI, Unix System Labora-
tories (USL), the AT&T subsidiary that held the Unix licence rights, tried to sue
first BSDI and then the University of California. The accusation was that the
company had distributed its intellectual property without permission. Follow-
ing various legal manoeuvres (which included a countersuit by the University
of California against USL), Novell bought the Unix rights from USL, and in
January 1994 reached an out-of-court settlement with the University of Cali-
fornia. As a result of this settlement, the CSRG distributed version 4.4BSD-Lite,
which was soon used by all the projects of the **BSD** family. Shortly afterwards
(after releasing version 4.4BSD-Lite Release 2), the CSRG disappeared. At that
point, some feared that it would be the end of **BSD** systems, but time has
shown that they are still alive and kicking under a new form of management
that is more typical of free software projects. Even in the first decade of the
year 2000 the projects managed by the **BSD** family are among the oldest and
most consolidated in the world of free software.

Bibliography

The history of Unix BSD is illustrative of a peculiar way of developing software during
the seventies and eighties. Whoever is interested in it can enjoy reading "Twenty years of
Berkeley Unix" (Marshall Kirk McKusick, 1999) [170], which follows the evolution from
the tape that Bob Fabry took to Berkeley with the idea of making one of the first versions
of Thompson and Ritchie's code function on a PDP-11 (bought jointly by the faculties of
informatics, statistics and mathematics), through to the lawsuits filed by AT&T and the
latest releases of code that gave rise to the **BSD** family of free operating systems.

2.2.3.   The beginnings of the Internet

Almost since its creation in the decade of the 1970s, Internet has been closely
related to free software. On the one hand, since the beginning, the communi-
ty of developers that built the Internet had several clear principles that would
later become classics in the world of free software; for example, the impor-
tance of users being able to help fix bugs or share code. The importance of
BSD Unix in its development (by providing during the eighties the most pop-
ular implementation of the TCP/IP protocols) made it easy to transfer many
habits and ways of doing things from one community - the developers centred
around the CSRG - to another community - the developers who were build-
ing what at the time was NSFNet and would later become Internet - and vice
versa. Many of the basic applications for the Internet's development, such as
Sendmail (mail server) or BIND (implementation of the name service) were
free and, to a great extent, the outcome of collaboration between these two
communities.

Finally, towards the end of the 80s and in the decade of the 90s, the free soft-
ware community was one of the first to explore in depth the new possibilities
offered by the Internet for geographically disperse groups to collaborate. To a
large extent, this exploration made the mere existence of the BSD community
possible, the FSF or the development of GNU/Linux.

One of the most interesting aspects of the Internet's development, from the free software point of view, was the completely open management of its documents and its rules. Although it may seem normal today (because it is customary, for example, in the IETF or the World Wide Web Consortium), at the time, the free availability of all its specifications, and design documents including the norms that define the protocols, was something revolutionary and fundamental for its development. In *Netizens. On the history and impact of Usenet and the Internet* [139] (page 106) we can read:

> "This open process encouraged and led to the exchange of information. Technical development is only successful when information is allowed to flow freely and easily between the parties involved. Encouraging participation is the main principle that made the development of the Net possible."

We can see why this paragraph would almost certainly be supported by any developer referring to the free software project in which he is involved.

In another quote, on "The evolution of packet switching" [195] (page 267) we can read:

> "Since ARPANET was a public project connecting many major universities and research institutions, the implementation and performance details were widely published."

Obviously, this is what tends to happen with free software projects, where all the information related to a project (and not only to its implementation) is normally public.

In this atmosphere, and before the Internet, well into the nineties, became an entire business, the community of users and its relationship with developers was crucial. During that period many organisations learned to trust not a single supplier of data communication services, but rather a complex combination of service companies, equipment manufacturers, professional developers, and volunteers, etc. The best implementations of many programs were not those than came with the operating system purchased together with the hardware, but rather free implementations that would quickly replace them. The most innovative developments were not the outcome of large company research plans but rather the product of students or professionals who tested ideas and collected feedback sent to them by various users of their free programs.

As we have already mentioned, Internet also offered free software the basic tools for long-distance collaboration. Electronic mail, news groups, anonymous FTP services (which were the first massive stores of free software) and, later, the web-based integrated development systems have been fundamental (and indispensable) for the development of the free software community as we know it today, and in particular, for the functioning of the immense majority of free software projects. From the outset, projects such as GNU or BSD

Bibliography

Readers interested in the evolution of the Internet, written by several of its key protagonists, can consult "A brief history of the Internet" (published by the ACM, 1997) [166].

made massive and intensive use of all these mechanisms, developing, at the same time as they used them, new tools and systems that in turn improved the Internet.

2.2.4. Other projects

During the 1980s many other important free software projects saw the light of day. We highlight for their importance and future relevance, X Window (windowing system for Unix-type systems), developed at the MIT, one of the first examples of large-scale funding for a free project financed by a business consortium. It is also worth mentioning Ghostscript, a PostScript document management system developed by a company called Aladdin Software, which was one of the first cases of searching for a business model based on producing free software.

Towards the end of the 1980s, there was already an entire constellation of small (and not so small) free software projects underway. All of them, together with the large projects we have mentioned up until now, established the bases of the first complete free systems, which appeared in the beginning of the 1990s.

2.3. Everything in its way

Around 1990, most of the components of a complete system were ready as free software. On the one hand, the GNU project and the BSD distributions had completed most of the applications that make up an operating system. On the other hand, projects such as X Window or GNU itself had built from windowing environments to compilers, which were often among the best in their class (for example, many administrators of SunOS or Ultrix systems would replace their system's proprietary applications for the free versions of GNU or BSD for their users). In order to have a complete system built exclusively with free software, just one component was missing: the kernel. Two separate and independent efforts came to fill the gap: 386BSD and Linux.

2.3.1. The quest for a kernel

Towards the end of the 1980s and beginning of the 1990s, the GNU project had a basic range of utilities and tools that made it possible to have a complete operating system. Even at the time, many free applications, including the particularly interesting case of X Window, were the best in their field (Unix utilities, compilers...). However, to complete the jigsaw a vital piece was missing: the operating system's kernel. The GNU project was looking for that missing piece with a project known as Hurd, which intended to build a kernel using modern technologies.

2.3.2. The *BSD family

Practically at the same time, the BSD community was also on the path towards a free kernel. The Net-2 distribution was only missing six files in order to complete it (the rest had already been built by the CSRG or its collaborators). In the beginning of 1992, Bill Jolitz finished those files and distributed 386BSD, a system that functioned on the i386 architecture and that in time would give rise to the projects NetBSD, FreeBSD and OpenBSD. Progress in the following months was fast, and by the end of the year it was sufficiently stable to be used in non-critical production environments, which included, for example, a windows environment thanks to the XFree project (which had provided X Window for the i386 architecture) or a great quality compiler, GCC. Although there were components that used other licences (such as those from the GNU projects, which used the GPL), most of the system was distributed under the BSD licence.

Bibliography

Some episodes of this period illustrate the capability of the free software development models. There is the well-known case of Linus Torvalds, who developed Linux while a second-year student at the University of Helsinki. But this is not the only case of a student who made his way thanks to his free developments. For example, the German Thomas Roel ported X11R4 (a version of the X Window system) to a PC based on a 386. This development took him to work at Dell, and later to become the founder of the X386 and XFree projects, which were fundamental for quickly giving GNU/Linux and the *BSDs a windows environment. You can read more about the history of XFree and Roel's role in it in "The history of xFree86" (*Linux Magazine*, December 1991) [135].

Then came the lawsuit from USL, which made many potential users fear proceedings against them in turn if the University of California were to lose the court case or simply, for the project to come to a standstill. Perhaps this was the reason why later, the installed base of GNU/Linux was much greater than all the *BSDs combined. But we cannot know this for sure.

2.3.3. GNU/Linux comes onstage

In July 1991 Linus Torvalds (a Finnish 21-year old student) placed his first message mentioning his project (at the time) to build a free system similar to Minix. In September he released the very first version (0.01), and every few weeks new versions would appear. In March 1994 version 1.0 appeared, the first one to be called *stable*, though the kernel that Linus built had been usable for several months. During this period, literally hundreds of developers turned to Linux, integrating all the GNU software around it, as well as XFree and many more free programs. Unlike the *BSDs, the Linux kernel and a large number of the components integrated around it were distributed with the GPL licence.

Bibliography

The story about Linux is probably one of the most interesting (and well-known) in the world of free software. You can find many links to information on it from the pages marking the 10th anniversary of its announcement, although probably one of the most interesting ones is the "History of Linux", by Ragib Hasan [138]. As a curiosity, you can consult

the thread on which Linus Torvalds announced that he was starting to create what later became Linux (in the newsgroup comp.os.minix) at http://groups.google.com/groups?th=d161e94858c4c0b9. There he explains how he has been working on his kernel since April and how he has already ported some GNU project tools onto it (specifically mentioning Bash and GCC).

Of the many developments to have emerged around Linux, one of the most interesting is the *distribution* concept[1]. The first distributions appeared soon, in 1992 (MCC Interim Linux, of the University of Manchester; TAMU, of Texas A&M, and the most well-known, SLS, which later gave rise to Slackware, which is still being distributed in the first decade of 2000), entailing the arrival of competition into the world of systems packaged around Linux. Each distribution tries to offer a ready-to-use GNU/Linux, and starting from the basis of the same software has to compete by making improvements considered important by their user base. In addition to providing pre-compiled ready-to-use packages, the distributions also tend to offer their own tools for managing the selection, installation, replacement and uninstallation of these packages, in addition to the initial installation on the computer, and the management and administration of the operating system.

> [1]This concept is explained in detail in the corresponding entry in Wikipedia, www.wikipedia.org/wiki/Linux_distribution

Over time, distributions have succeeded each other as different ones became the most popular. Of them all, we would highlight the following:

1) Debian, developed by a community of volunteer users.

2) Red Hat Linux, which was first developed internally by the company Red Hat, but which later adopted a more community-based model, giving rise to Fedora Core.

3) Suse, which gave rise to OpenSUSE, following a similar evolution to Red Hat.

4) Mandriva (successor of Mandrake Linux and Conectiva).

5) Ubuntu, derived from Debian and produced on the basis of Debian by the company Canonical.

2.4.   A time of maturation

Midway through the first decade of 2000, GNU/Linux, OpenOffice.org or Firefox were present in the media quite often. The overwhelming majority of companies use free software for at least some of their IT processes. It is difficult to be an IT student and not to use large amounts of free software. Free software is no longer a footnote in the history of IT and has become something very important for the sector. IT companies, companies in the secondary sector (those that use software intensively, even though their primary activity is different)

and public administrations are starting to consider it as something strategic. And slowly but surely it is arriving among domestic users. In broad terms, we are entering a period of maturation.

And at the bottom of it all, an important question starts to arise, which summarises in a way what is happening: "are we facing a new model of software industry?". Perhaps, it may yet happen that free software becomes no more than a passing trend to be remembered nostalgically one day. But it may also be (and this seems increasingly likely) a new model that is here to stay, and perhaps to change radically one of the youngest but also most influential industries of our time.

2.4.1. End of the nineties

In the mid-1990s, free software already offered complete environments (distributions of GNU/Linux, *BSD systems...) that supported the daily work of many people, especially software developers. There were still many pending assignments (the main one to have better graphical user interfaces at a time when Windows 95 was considered the standard), but there were already several thousand people worldwide who used exclusively free software for their day to day work. New projects were announced in rapid succession and free software embarked on its long path towards companies, the media and public awareness in general.

This period is also associated with Internet taking off as a network for everyone, in many cases led by the hand of free programs (especially in its infrastructure). The net's arrival into the homes of millions of end users consolidated this situation, at least in terms of servers: the most popular web (HTTP) servers have always been free (first the NCSA server, followed by Apache).

Perhaps the beginning of the road for free software until its full release among the public is best described in the renowned essay by Eric Raymond, "The cathedral and the bazaar" (Eric S. Raymond, 2001) [192]. Although much of what is described in it was already well known by the community of free software developers, putting it into paper and distributing it extensively made it an influential tool for promoting the concept of *free software* as an alternative development mechanism to the one used by the traditional software industry. Another important paper of this period was "Setting up shop. The Business of open source software" [141], by Frank Hecker, which for the first time described the potential business models for free software, and which was written in order to influence the decision to release the Netscape Navigator code.

Whereas Raymond's paper was a great tool for promoting some of the fundamental characteristics of free software, the release of Netscape Navigator's code was the first case in which a relatively large company, in a very innovative sector (the then nascent web industry) made the decision to release one of its products as free software. At that time, Netscape Navigator was losing the web

navigators' battle against Microsoft's product (Internet Explorer), partly due to Microsoft's tactics of combining it with its operating system. Many people believe that Netscape did the only thing that it could have done: to try to change the rules to be able to compete with a giant. And from this change in the rules (trying to compete with a free software model) the Mozilla project was born. This project, which had its own problems, has led several years later to a navigator that, although it has not recovered the enormous market share that Netscape had in its day, seems technically at least as good as its proprietary competitors.

In any case, irrespective of its later success, Netscape's announcement that it would release its navigator's source code had a great impact on the software industry. Many companies started to consider free software worthy of consideration.

The financial markets also started paying attention to free software. In the euphoria of the dotcom boom, many free software companies became targets for investors. Perhaps the most renowned case is that of Red Hat, one of the first companies to realise that selling CDs with ready-to-use GNU/Linux systems could be a potential business model. Red Hat started distributing its Red Hat Linux, with huge emphasis (at least for what was common at the time) on the system's ease of use and ease of maintenance for people without a specific IT background. Over time it diversified, keeping within the orbit of free software, and in September 1998 it announced that Intel and Netscape had invested in it. "If it is good for Intel and Netscape, it must be good for us", is what many investors must have thought then. When Red Hat went public in summer 1999, the IPO was subscribed completely and soon the value of each share rose spectacularly. It was the first time that a company was obtaining financing from the stock exchange with a model based on free software. But it was not the only one: later, others such as VA Linux or Andover.net (which was later acquired by VA Linux) did the same.

Note

Red Hat provides a list of its company milestones at http://fedora.redhat.com/about/history/.

During this period, many companies were also born with business models based on free software. Despite not going public or achieving such tremendous market caps, they were nevertheless very important for the development of free software. For example, many companies appeared that started distributing their own versions of GNU/Linux, such as SuSE (Germany), Conectiva (Brazil) or Mandrake (France), which would later join the former in order to create Mandriva. Others offered services to companies that wanted maintenance or to adapt free products: LinuxCare (US), Alcove (France), ID Pro (Germany), and many more.

Meanwhile, the sector's giants started to position themselves in relation to free software. Some companies, such as IBM, incorporated it directly into their strategy. Others, such as Sun Microsystems, had a curious relationship with it, at times backing it, at others indifferent, and at others confrontational. Most (such as Apple, Oracle, HP, SGI, etc.) explored the free software model with various strategies, ranging from the selective freeing of software to straightforward porting of their products to GNU/Linux. Between these two extremes there were many other lines of action, such as the more or less intensive use of free software in their products (such as the case with Mac OS X) or the exploration of business models based on the maintenance of free software products.

From the technical point of view, the most remarkable event of this period was probably the appearance of two ambitious projects designed to carry free software to the *desktop* environment for inexperienced IT users: KDE and GNOME. Put simplistically, the final objective was not to have to use the command line in order to interact with GNU/Linux or *BSD or with the programs on those environments.

KDE was announced in October 1996. Using the Qt graphic libraries (at that time a proprietary product belonging to the company Trolltech, but free of charge for use on GNU/Linux[2]), construction began of a set of desktop applications that would work in an integrated manner and have a uniform appearance. In July 1998 version 1.0 of the K Desktop Environment was released, and was soon followed by increasingly more complete and more mature new versions. GNU/Linux distributions soon incorporated KDE as a desktop for their users (or at least as one of the desktop environments that users could choose).

[2]Later, Qt started to be distributed under the free licence QPL (Qt Public Licence), non-compatible with GPL, which caused some problems, since most of KDE was distributed under the GPL. In time, Trolltech finally decided to distribute Qt under the GPL licence, bringing these problems to an end.

Mostly as a reaction to KDE's dependence on the Qt proprietary library, in August 1997 the GNOME project was announced (Miguel de Icaza, "The story of the GNOME Project") [101], with similar goals and characteristics to those of KDE, but stating the explicit objective of all its components being free software. In March 1999, GNOME 1.0 was released, which would also improve and stabilise over time. As of that moment, most distributions of free operating systems (and many Unix-derived proprietary ones) offered the GNOME or KDE desktop as an option, and the applications of both environments.

Meanwhile, the main free software projects underway remained in good health with new projects emerging almost every day. In various niche markets, free software was found to be the best solution (acknowledged almost worldwide). For example, since its appearance in April 1995, Apache has maintained the largest market share for web servers; XFree86, the free project that develops X Window, is by far the most popular version of X Window (and therefore, the most extended windows system for Unix-type systems); GCC is recognised as the most portable C compiler and one of the best quality; GNAT, the compilation system for Ada 95, has conquered the best part of the market for Ada compilers in just a few years; and so on.

In 1998, the Open Source Initiative (OSI) was founded, which decided to adopt the term *open source software* as a brand for introducing free software into the business world, while avoiding the ambiguity of the term *free* (which can mean both free to use and free of charge). This decision sparked one of the fiercest debates in the world of free software (which continues to this day), since the Free Software Foundation and others considered that it was much more appropriate to speak about *free software* (Richard Stallman, "Why *free software* is better than *open source*", 1998) [206]. In any case, the OSI made a great promotional campaign for its new brand, which has been adopted by many as the preferred way to talk about free software, especially in the English-speaking world. To define *open source* software, the OSI used a definition derived from the one used by the Debian project to define free software ("Debian free software guidelines", http://www.debian.org/social_contract.html#guidelines) [104], which at the same time fairly closely reflects the idea of the FSF in this regard ("Free software definition", http://www.gnu.org/philosophy/free-sw.html) [120], meaning that from the practical point of view almost any program considered free software can also be considered *open source* and vice versa. However, the free software and open source software communities (or at least the people who identify with them) can be very different.

2.4.2. Decade of 2000

In the early years of the decade of 2000, free software was already a serious competitor in the servers segment and was starting to be ready for the desktop. Systems such as GNOME, KDE, OpenOffice.org and Mozilla Firefox can be used by domestic users and are sufficient for the needs of many companies, at least where office applications are concerned. Free systems (and especially systems based on Linux) are easy to install, and the complexity of maintaining and updating them is comparable to that of other proprietary systems.

Right now, every company in the software industry has a strategy with regards to free software. Most of the leading multinationals (IBM, HP, Sun, Novell, Apple, Oracle...) incorporate free software to a greater or lesser extent. At one extreme we can find companies such as Oracle, which react by simply porting their products to GNU/Linux. At another extreme, we can find IBM, which has the most decisive strategy and has made the biggest publicity campaigns about GNU/Linux. Among the leaders in the IT market, only Microsoft has positioned itself in clear opposition to free software and particularly software distributed under the GPL licence.

As regards the world of free software itself, despite the debates that occasionally stir the community, its growth is massive. Every day there are more developers, more active free software projects, more users, etc. With each passing day free software is moving away from the sidelines and becoming a force to be reckoned with.

In light of this, new disciplines are emerging that specifically study free software, such as free software engineering. Based on research, bit by bit we are starting to understand how free software operates in its various aspects: development models, business models, coordination mechanisms, free project management, developers' motivations, etc.

These years we are also starting to see the first effects of the *offshoring* that free software development allows: countries considered "peripheral" are actively participating in the world of free software. For example, the number of Mexican or Spanish developers (both countries with a limited tradition of software industry) in projects such as GNOME is significant (Lancashire, "Code, culture and cash: the fading altruism of open source development", 2001) [164]. And the role of Brazil is even more interesting, with its numerous developers and experts in free software technologies, and decisive backing from the public administrations. gnuLinEx is a case that merits special attention, as an example of how a region with very little tradition of software development can try to change the situation through an aggressive strategy of free software implantation.

From the decision-making perspective when it comes to implementing software solutions, we would highlight the fact that there are certain markets (such as Internet services or office applications) in which free software is a natural choice that cannot be overlooked when studying what type of system to use.

On the negative front, these years have seen how the legal environment in which free software operates is changing rapidly worldwide. On the one hand, software patents are increasingly adopted in more and more countries. On the other hand, new copyright laws make it difficult or impossible to develop free applications in some spheres, the most well-known one being DVD viewers (due to the CSS encoding algorithm that this technology uses).

gnuLinEx

In the beginning of 2002 the Extremadura Regional Government publicly announced the gnuLinEx project. The idea was simple: to promote the creation of a distribution based on GNU/Linux with the fundamental objective of using it on the thousands of computers to be installed in public schools throughout the region. Extremadura, situated in the western part of Spain, bordering Portugal, has approximately 1 million inhabitants and has never stood out for its technological initiatives. In fact, the region had practically no software industry.

In this context, gnuLinEx has made a very interesting contribution to the free software panorama on a global scale. Beyond being just a new distribution of GNU/Linux based on Debian (which is still a worthy anecdote), and beyond its enormous impact on the mass media (it was the first time that Extremadu-

ra made the front cover of *The Washington Post* and one of the first that a free software product did), what is extraordinary is the (at least apparently) solid backing of a public administration for free software. The Regional Government of Extremadura decided to try a different model where educational software was concerned, and then to extend this model to all the software used within the scope of its influence. This has made it the first public administration of a developed country to have decisively adopted this approach. A lot of interest was generated around the Regional Government's initiative, within Extremadura and outside of it: there are academies that teach IT using gnuLinEx; books have been written to support this teaching; computers are being sold with gnuLinEx pre-installed. In general, they are trying to create an educational and business fabric around this experience in order to give it support. And the experience has been exported. At the beginning of the 21st century, several autonomous communities in Spain have backed free software in education (in one way or another), and in general, its importance for public administrations is widely acknowledged.

Knoppix

Since the end of the nineties, there are GNU/Linux distributions that can be easily installed, but Knoppix, whose first version appeared in 2002, has probably allowed this idea to reach its full expression. It is a CD that boots on almost any PC, converting it (without even having to format the disk, since it can be used "live") into a fully functional GNU/Linux machine, with a selection of the most frequent tools. Knoppix combines good automatic hardware detection with a good choice of programs and "live" functioning. For example, it allows a rapid and direct experience of what it means to work with GNU/ Linux. And it is giving rise to an entire family of distributions of the same type, specialised for the specific requirements of a user profile.

OpenOffice.org

In 1999, Sun Microsystems bought a German company called Stardivision, whose star product was StarOffice, a suite of office applications similar in functionality to the Microsoft Office set of tools. One year later, Sun distributed most of the StarOffice code under a free licence (the GPL) creating the OpenOffice.org project. This project released version 1.0 of OpenOffice.org in May 2002. OpenOffice.org has become a quality suite of office applications with a similar functionality to that of any other office product, and, more importantly, it interoperates very well with the Microsoft Office data formats. These features have made it the reference free software application in the world of office suites.

The importance of OpenOffice.org, from the point of view of extending free software to a large number of users, is enormous. Finally it is possible to change, almost without problems, from the proprietary environments common with office suites (undoubtedly the star application in the business

world) to totally free environments (such as GNU/Linux plus GNOME and/or KDE plus OpenOffice.org). Also, the transition can be made very smoothly: since OpenOffice.org also works on Microsoft Windows, it is not necessary to change operating systems in order to experiment in depth with using free software.

Mozilla, Firefox and the rest

Practically since its appearance in 1994 until 1996, Netscape Navigator was the unchallenged market leader in web navigators, with market shares of up to 80%. The situation started to change when Microsoft included Internet Explorer with Windows 95, causing Netscape Navigator to gradually lose market share. At the beginning of 1998 Netscape announced that it was going to distribute a large part of its navigator code as free software, which it did in March that same year, launching the Mozilla project. For quite a while the project was clouded by uncertainty, and even pessimism (for example, when its leader, Jamie Zawinski, abandoned it), because as time went by no product was resulting from its launch.

In January 2000, the project released Mozilla M13, which was considered the first relatively stable version. But in just May 2002 version 1.0 was finally published, the first officially stable version, over four years after the first Netscape Navigator code had been released.

> **Bibliography**
>
> In "Netscape Navigator", by Brian Wilson, [234], we can consult a detailed list of the main versions of Netscape Navigator and Mozilla, and their main characteristics.

Finally Mozilla had become a reality, although perhaps too late, if we bear in mind the market shares that Internet Explorer had in 2002 or 2003 (when it was the undisputed leader leaving Mozilla and others in a totally marginal position). But despite taking so long, the Mozilla project has borne fruit; not only expected fruit (the Mozilla navigator), but also other "collateral" ones, such as Firefox for example, another navigator based on the same HTML engine, which has become the main product, and which since it appeared in 2005 is managing bit by bit to erode other navigators' market share.

The Mozilla project has helped to fill a large gap in the world of free software. Before Konqueror appeared (the KDE project's navigator), there were not many free navigators with a graphic interface. Since the publication of Mozilla, an enormous number of projects based on it have emerged which have produced a large number of navigators. At the same time, the combination of Mozilla Firefox and OpenOffice.org allows free software to be used for the most common tasks, even in a Microsoft Windows environment (they both work not only on GNU/Linux, *BSD and other Unix-type systems, but also on Windows). For the first time in the history of free software, it has made the transition from proprietary software to free software in office environments a simple task: we can start by using these two applications on Windows, without changing operating systems (for those who use it normally), and over time eliminate the only non-free part and move onto GNU/Linux or FreeBSD.

The case of SCO

At the beginning of 2003, the SCO corporation (formerly Caldera Systems and Caldera International) presented a legal case against IBM for alleged breach of its intellectual property rights. Although the case was complex, it centred on the accusation that IBM had contributed to the Linux kernel with code belonging to SCO. In May 2007, the matter had still not been resolved and had even become more complicated by further legal suits (IBM and Red Hat against SCO, SCO against AutoZone and DaimlerChrysler, two large IT users) and by SCO's campaigns threatening to pursue big companies that used Linux, etc.

Although the winner of this enormous legal battle has still not emerged, the case has highlighted certain legal aspects concerning free software. In particular, many companies have considered the problems that they may have to face if they use Linux and other free programs, and the guarantee that in doing so they are not in breach of third party intellectual or industrial property rights.

In some way, this case and other ones (such as those related to the validity of the GPL licences which were resolved in Germany in 2005) may also be interpreted as a sign of the maturity of free software. It has stopped being a stranger to the business world to become part of many of its activities (including those related to legal strategies).

Ubuntu, Canonical, Fedora and Red Hat

Although Canonical (the company that produces and distributes Ubuntu) could be considered a recent arrival to the business of GNU/Linux distributions, its activities deserve our attention. In a relatively short time, Ubuntu has established itself as one of the best known and most widely used distributions, with a reputation for good quality, and great ease of installation and use. Ubuntu also stands out for its greater attention to including fundamentally free software than most distributions produced by companies.

However, the fundamental characteristic of Ubuntu (and of Canonical's strategy) has been to base on Debian, a distribution created and maintained by volunteers. In fact, Ubuntu is not the first case of a distribution based on Debian (another well-known case is gnuLinEx), but perhaps it is the one to have received the most funding. For example, Canonical has hired a large number of Debian experts (many of whom participate in the project) and has pursued a strategy that seeks collaboration with the volunteer project. To some extent, Canonical has tried to fill what it considers is missing from Debian in order to gain acceptance from the average user.

Red Hat, in turn, has followed a different path in order to wind up in a fairly similar situation. Starting from a distribution produced entirely with its own resources, it decided to collaborate with Fedora, a group of volunteers that was already working with distributions based on Red Hat, in order to produce

Fedora Core, its "popular" distribution. Red Hat maintains its version for companies, but this collaboration with volunteers is, in the end, very similar to the one that has produced Ubuntu.

Perhaps all of these movements are no more than the product of the fierce competition taking place in the market for GNU/Linux distributions and of one more notable trend: companies' collaboration with volunteers (with *the community*) to produce free software.

Customised distributions

Since Linux came onto the scene, a large number of groups and companies have created their own distributions based on it. But during these years, the phenomenon has caught on with many organisations and companies that want customised versions for their own requirements. Customisation has been able to expand because the process has become cheaper and there is widespread availability of the technical knowledge to do so, even making this a niche market for certain companies.

Perhaps one of the best known cases of customised distributions is the one for Spain's autonomous communities. The Extremadura Regional Government with its gnuLinEx sparked a trend that many other autonomous communities have since followed. The process is so common that several of them regularly convene tenders for the creation and maintenance of new versions of their distributions.

The creation of customised distributions realises a trend that the world of free software had been discussing for a long time: adapting programs to users' specific needs without it having to be the original producers that necessarily make the adaptation.

Bibliography

Some of the most well-known distributions of GNU/Linux in the Spanish autonomous communities include:

- gnuLinEx: http://linex.org (Extremadura)

- Guadalinex: http://guadalinex.org (Andalucía)

- Lliurex: http://lliurex.net (Comunidad Valenciana)

- Augustux: http://www.zaralinux.org/proy/augustux/ (Aragón)

- MAX: http://www.educa.madrid.org/web/madrid_linux/ (Madrid)

- MoLinux: http://molinux.info (Castilla-La Mancha)

Company-company and volunteer-company collaborations

Since practically the beginning of free software, there have been companies that collaborated with volunteers in developing applications. However, in these years when it appears that we are reaching maturity there is a growing

number of companies that use free software as part of their strategy to collaborate with other companies, when they find it interesting. Two of the most significant cases, organised specifically with this objective, are ObjectWeb (an alliance formed in France which over time clearly has clearly become international) and Morfeo (in Spain). In both cases, a group of companies has agreed to develop a set of free systems that are of interest to them, and decided to distribute it as free software.

In other cases, companies have actively sought to collaborate in free projects promoted by volunteers, or tried to make volunteers collaborate with their own free projects. The GNOME Foundation or the already-mentioned Ubuntu in respect of Debian are examples of this first scenario. Sun and OpenOffice.org and OpenSolaris, or Red Hat with Fedora Core, are examples of the second.

Expanding to other spheres

Free software has proven that in the field of producing programs there is another way of doing things. In practice, we have seen how granting the freedom to distribute, modify and use can achieve sustainability, either through volunteer work, or through business generation that allows companies to survive.

As time passes, this same idea is being transferred to other spheres of intellectual work. The Creative Commons licences have made it possible to free spheres such as literature, music, or video. Wikipedia is proving that a field as particular as the production of encyclopaedias can move through a very interesting path. And there are more and more literary authors, music bands and even film producers interested in models of free production and distribution.

In all these domains there is still a long way to go, and in almost all of them practice has not yet fully proven that sustainable creation is possible with free models. But it cannot be denied that experimentation with it is reaching a boiling point.

Free software as a subject of study

Although some works, such as the renowned "The cathedral and the bazaar" cleared the way for the study of free software as such, it was not until 2001 and subsequent years that the academic community started to consider free software as something worthy of study. Over time, the massive availability of data (almost everything in the world of free software is public and available from public information archives) and the innovations that free software has provided have drawn the attention of many groups. Midway through the decade of 2000 there are already several international conferences centred specifically on free software, top-ranking magazines frequently produce papers on it, and research-funding agencies are opening lines aimed specifically towards it.

2.5. The future: an obstacle course?

Of course, it is difficult to predict the future. And that is certainly not our objective. Therefore, rather than trying to explain what the future of free software will be like, we will try to show the problems that it will foreseeably have to face (and has indeed been facing for a long time). How the world of free software is able to overcome these obstacles will undoubtedly determine its situation in several years' time.

- FUD (*fear, uncertainty, doubt*). This is a fairly common technique in the world of information technologies, used by free software's competitors in order to discredit free software, with more or less justification and varying degrees of success. In general terms, free software has been fairly immune to these techniques, perhaps due to its complexity and different ways of seeping into companies.

- Dissolution. Many companies are testing the limits of free software as a model, and in particular are trying to offer their clients models that present some similar characteristics to free software. The main problem that can present itself with this type of model is that it generates confusion among clients and developers, who need to read the small print in detail in order to realise that what they are being offered does not have the advantages that free software offers them. The most well-known model of this type is the Shared Source program, by Microsoft.

- Lack of knowledge. In many cases, users turn to free software simply because they think that it is free of charge; or because they think that it is "fashionable". If they do not look deeper into it, and study with a certain amount of detail the advantages that free software can offer as a model, they run the risk of not taking full advantage of them. In many cases, the initial assumptions in the world of free software are so different from the traditional ones in the world of proprietary software that a minimum analysis is required in order to understand that what in one case is frequent in the other may be impossible, and vice versa. Therefore, lack of knowledge can only generate dissatisfaction and loss of opportunities for any person or organisation approaching free software.

- Legal obstacles. This is certainly the main problem that free software is going to have to deal with in coming years. Although the legal environment in which free software developed in the 80s and first half of the 90s was not ideal, at least it left enough space for it to grow freely. Since then, extension of the scope of patenting to software (which has occurred in many developed countries) and new copyright legislation (limiting the software developer's liberty to create) are producing increasingly higher barriers to free software's entry into important segments of applications.

2.6. Summary

This chapter presents the history of free software. The sixties was a period dominated by large computers and IBM in which software was distributed together with the hardware, and usually with the source code. In the seventies, software started to be sold separately, and soon proprietary distributions, which did not include source code and did not give permission to modify or redistribute, became almost the only option.

In the decade of the 1970s, work began on developing the Unix operating system at AT&T's Bell Labs, giving rise later to Unix BSD. Its evolution, in parallel with the birth of the Internet, served as a testing field for new ways of developing in collaboration, which later became common in the world of free software.

In 1984, Richard Stallman started to work on the GNU project, founding the Free Software Foundation (FSF), writing the GPL licence, and in general establishing the foundations of free software as we now know it.

In the 90s Internet matured offering free software communities new channels for communication and distribution. In 1991, Linus Torvalds started to develop a free kernel (Linux) which helped to complete the GNU system, which already had almost all the parts for becoming a complete system similar to Unix: C compiler (GCC), editor (Emacs), windowing system (X Window), etc. This is how the GNU/Linux operating systems were born, branching out into many distributions, such as Red Hat Linux and Debian GNU/Linux. Towards the end of the 90s, these systems were completed with two desktop environments: KDE and GNOME.

In the decade of 2000, free software managed to lead in some sectors (such as for web servers, dominated by Apache), and new tools appeared covering a large number of IT requirements.

See also

Interested readers will find in Appendix B a list of some of the most relevant dates in the history of free software.

3. Legal aspects

"The licences for most software are designed to take away your freedom to share and change it."

GNU General Public Licence, version 2

This chapter looks at the main legal aspects related to free software. To put them into context, we start with a small introduction to the most basic concepts of intellectual and industrial property rights, before offering the detailed definition of *free software*, *open source software* and other related concepts. We also look in some detail at the most common free software licences and their impact on business models (subject covered in greater detail in chapter 5) and development models.

3.1. Brief introduction to intellectual property

The term *intellectual property* has various meanings according to its context and who uses it. Nowadays it is frequently used in many spheres to refer to various privileges awarded over intangible goods with economic value. It includes concepts such as *copyright* and similar, which protect from unauthorised copy literary or artistic works, computer programs, data compilations, industrial designs, etc.; trademarks, which protect symbols; geographical indications, which protect appellations of origin; trade secrets, which protect the hiding of information, and patents, which concede temporary monopolies to inventions in exchange for their revelation. However, in many legal traditions, including the Hispanic tradition, a distinction is made between *intellectual property*, which refers exclusively to copyright, and *industrial property,* which covers the other concepts.

In any case, the legislation applicable to all of these aspects is one of the best coordinated practically worldwide. On the one hand, the WIPO (Worldwide International Property Organisation) covers both types of property in all of their aspects. On the other hand, the TRIPS agreement (Trade-Related aspects of Intellectual Property rights) establishes certain minimum levels of protection and obliges all member countries of the WTO (World Trade Organisation) to develop them within certain timeframes, according to the level of development of the country.[3]

[3]The TRIPS agreement was signed under pressure from the industrialised countries (especially the US and Japan).

Article 27 of the Declaration of Human Rights acknowledges that everyone has the right to the protection of the moral and material interests resulting from any scientific, literary or artistic production of which he is the author. However, in many cases (and frequently in the case of software), this right is transferred in practice to the companies that employ the creators or that distribute or sell their creations. Nonetheless, intellectual property is justified

not just morally, but also for practical reasons, in order to comply with another right: the public's right to benefit from creation, promoting it through incentives and protecting investments in creation, research and development. In order to harmonise these two rights, intellectual property is temporary and expires once it has fulfilled its function of promotion.

But expiry is not the only distinguishing feature between intellectual property and ordinary property. Nowadays, its products can be copied easily and cheaply, without any loss of quality. Copying does not prejudice the party that is already benefiting from what is copied, unlike theft, which does deprive the original possessor. Copying can prejudice the owner, by depriving him of potential income from a sale. Controlling the copying of intangibles is much more complicated than controlling the theft of tangible property and can lead us to a situation of a police State, having to control all copies of information, and legal insecurity, since the potential for accidental infringement of rights increases. Furthermore creativity is incremental: creating always copies something, and the dividing line between a poor imitation and inspiration is a subtle one.

In order to study this in more depth, the following sections go over some of the categories of intellectual property. In any case, we can already advance that free software proposes a new point of equilibrium in this sphere, advocating the benefits of copying and incremental innovation versus exclusive control of a work by its author.

3.1.1. Copyright

Copyright protects the expression of a content, not the content itself. Copyright was developed in order to compensate the authors of books or art. Protected works may express ideas, knowledge or methods that are freely usable, but it is prohibited to reproduce them without full or partial permission, with or without modifications. This protection is very simple, since it automatically comes into force with an almost universal scope just when the work is published/released. Currently, it has been extended to computer programs and (in some geographical areas) to data compilations.

The Law on Intellectual Property (LPI) in Spain, and similar laws in other countries, developed on the basis of the Berne Convention of 1886 for the protection of literary and artistic works, regulates copyright. These rights are divided into moral and intellectual rights. The former guarantee the author's control over the distribution of his work, under his name or pseudonym, the recognition of authorship, respect for the integrity of the work and the right to modify and withdraw it. The second give the author the right to exploit the work economically and may be ceded in whole or in part, exclusively or not, to a

third party. Moral rights are lifelong or indefinite, whereas intellectual rights have a fairly long duration (seventy years following the author's death, in the case of a physical person and Spanish law).

Cession of these rights is established by means of a contract known as a *licence*. In the case of proprietary programs, these are generally distributed through "non exclusive" licences for use, understood as automatically accepted by opening or installing the product. Therefore it is not necessary to sign the contract, since in the case of the receiver not accepting it, the rights by default under the law govern automatically, that is none. Licences cannot restrict some of the rights granted by current legislation, such as the right to make private copies of art or music, which allows a copy of a recording to be given to a friend as a gift, but this right does not apply to programs. According to the LPI of 1996 (Spanish Law on Intellectual Property. Royal Legislative Decree 1/1996, of 12th April) [77], modified in 2006 (Law on Intellectual Property. Law 23/2006, of 7th July) [79], in respect of programs it is always possible to make a backup copy, they may be studied for making programs interoperable and they may be corrected and adapted to our needs (which is difficult, because normally we do not have access to the source code). These rights may not be restricted through licences, although the laws are under review, in an apparently unstoppable trend to limit the rights of users. Organised compilations of works or third party data are also subject to copyright, although under different terms with a shorter timeframe.

New information technologies, and specially the web, have deeply transformed copyright protection, since expressions of content are much easier to copy than content itself. And in the case of programs and some works of art (music, images, films, and even literature) they "work" automatically on the computer without the need for any appreciable human effort. However, designs or inventions need to be built and possibly put into production. This possibility of generating wealth at no cost has led a large proportion of the public, in particular in poor countries, to duplicate programs without paying the licence, without public awareness of this being a "malicious action" (unlike in the case of stealing physical property, for example). Meanwhile, program manufacturers, either alone or in coalition (through the BSA, Business Software Alliance, for example), exert enormous pressure for licences to be paid and for governments to pursue what has become known as *piracy*.

Note

The word *piracy* has become generally accepted as a synonym for the 'violation of any form of intellectual property, especially in the case of illegally copying of programs, music and films'. The term seems exaggerated and in the dictionary of the Royal Spanish Academy of Language it appears with that meaning in the figurative sense, since the original word refers to 'robbery with violence committed at sea'. This is why Richard Stallman recommends avoiding it ("Some confusing or loaded words and phrases that are worth avoiding", 2003) [212].

Precisely in order to protect the copyright of contents with proprietary licences, the so-called DRM systems were born (*digital rights management*), designed to control access and the use of data in digital format or to restrict its use to certain devices. The use of DRM systems has been strongly criticised in many sectors, because they protect copyright imposing restrictions beyond what is sufficient, which is why some, such as the Free Software Foundation, recommend interpreting the acronym as *digital restrictions management*, in an attempt to avoid using the word *rights*), because it considers that there is an excessive deprivation of the rights of users in favour of satisfying copyright demands.

3.1.2. Trade secret

Another resource that companies use in order to make profit from their investments is trade secret, protected by the laws of industrial property, provided that companies take sufficient measures to hide the information that they do not wish to disclose. In the case of chemical or pharmaceutical products that require governmental approval, the State undertakes not to disclose submitted data that is not mandatory to make public.

One of the best known applications of trade secret is the propietary software industry, which generally sells compiled programs without access to the source code, in order to prevent derived programs from being developed.

At first sight it would appear that the protection of trade secret is perverse, since it can deprive society of useful knowledge indefinitely. To some extent some legislations also interpret it this way, and allow reverse engineering in order to develop replacement products, although industry pressure has managed to prohibit this activity in many countries, and in other countries only made it possible on the grounds of compatibility.

Whether or not trade secret is a perversion, in many cases it is better than a patent since it offers a competitive advantage to the person placing a product on the market while the competition tries to imitate it through reverse engineering. The more sophisticated the product the more it will cost the competition to reproduce it, whereas if it is trivial, it will be copied quickly. Imitation with improvements has been fundamental in the development of today's superpowers (the US and Japan) and is very important for the financial independence of developing countries.

3.1.3. Patents and utility models

The alternative to trade secret is a patent. In exchange for a seventeen to twenty five year monopoly and a specific financial cost, an *invention* is publicly disclosed so that it can be easily reproduced. It aims to promote private re-

search, at no cost to the taxpayer and without losing the outcome. The patent holder can decide whether to allow others to use it and the price to be paid for the licence.

Official doctrine is that the patent system promotes innovation, but more and more voices are making themselves heard with the view that it impedes it, either because the system is poorly implemented or because they consider that it is perverse in itself (François-René Rideau, "Patents are an economic absurdity", 2000) [194].

What is considered an invention has changed over time, and there is enormous pressure to extend the scope of the system, to include algorithms, programs, business models, natural substances, genes and forms of life, including plants and animals. TRIPS requires the patents system not to discriminate against any field of knowledge. The pressures of the World Intellectual Property Organisation (WIPO or OMPI) aim to eliminate the need for an invention to have an industrial application and also to reduce the standards of invention required of a patent. The US is at the forefront of countries with minimum requirements on patentability, and is also the most belligerent for other countries to adopt its standards, forgetting that the US refused to accept foreign patents when it was an underdeveloped country.

After obtaining a patent, the rights of the owner are independent of the quality of the invention and the effort invested in obtaining it. Given the cost of maintaining a patent, and litigation costs, only large companies are able to maintain and do maintain a large portfolio of patents, which puts them in a position to strangle any competition. Given the ease of getting patents on trivial or widely applicable solutions, they can thus monopolise an extensive field of economic activity.

With patents, many activities, especially programming, become extremely risky, because it is very easy that in developing a complicated program there is an accidental violation of some patent. When two or more companies are conducting research in order to resolve a problem, it is highly probable that they will reach a similar solution at almost the same time, but only one of them (usually the one with most resources) will manage to patent its invention, preventing the others from having any chance of recouping their investment. Any complex technological development becomes a nightmare if in order to solve each part you first need to find out whether the solution found is patented (or patent pending), so as to obtain the licence or find an alternative solution. This problem is particularly severe with free software, where the violation of algorithm patents is evident from simply inspecting the code.

Although in Europe it is still illegal to patent an algorithm, it will become possible in the near future, perhaps by the time the reader reads these lines.

3.1.4. Registered trademarks and logos

Trademarks and logos are names and symbols that represent an established quality (or a massive investment in publicity). They are not very important in the world of free software, possibly because registering them has a cost. Therefore, just a few important names such as Open Source (by the Open Source Foundation), Debian (by Software in the Public Interest), GNOME (by the GNOME Foundation), GNU (by the Free Software Foundation) or OpenOffice.org (by SUN Microsystems) are registered, and only in a few countries. However, not registering the names has caused problems. For example, in the US (1996) and in Korea (1997) people have registered the name Linux and demanded payment for its use. Resolving these disputes entails legal costs and the need to prove the use of the name prior to the date of registration.

3.2. Free software licences

Legally speaking, the situation of free programs in relation to proprietary ones is not very different: they are both distributed under a licence. The difference lies in what the licence allows. In the case of free program licences, which do not restrict particularly their use, redistribution and modification, what can be imposed are conditions that need to be met precisely in the case of wanting to redistribute the program. For example, it is possible to demand observation of authorship indications or to include the source code if wanting to redistribute the program ready to run.

Although essentially *free software* and *propietary software* differ in terms of the licence under which the authors publish their programs, it is important to emphasise that this distinction is reflected in completely different conditions of use and redistribution. As we have seen in the last few years, this has not only given rise to totally different development methods, but also to practically opposite ways (in many aspects) of understanding IT.

The laws on intellectual property ensure that in the absence of explicit permission virtually nothing can be done with a work (in our case, a program) received or purchased. Only the author (or the holder of the rights to the work) can grant us that permission. In any case, ownership of the work does not change by granting a licence, since this does not entail transfer of ownership, but rather just the right of use, and in some cases (mandatory with free software), of distribution and modification. Free software licences are different from proprietary software licences precisely in that instead of carefully restricting what is allowed, it makes certain explicit allowances. When people receive a free program they may redistribute it or not, but if they do redistribute it, they can only do so because the licence allows it. But to do so the licence must be observed. Indeed, the licence contains the rules of use that users, distributors, integrators and all other parties involved in the world of IT must observe.

In order to fully understand all the legal ins and outs that arise in this chapter (and which are without question very important to understanding the nature of free software) we should also be aware that each new version of a program is considered a new work. The author, once again, is fully entitled to do what he wants with it, even to distribute it under totally different terms and conditions (in other words, with a totally different licence to the earlier one). That way if the reader is the sole author of a program, he may publish one version under a free software licence and, if he wishes to, a later one under a proprietary licence. In the case of there being more authors and the new version containing code that they have produced, if it will be published under different conditions, all of them will have to approve the change in licence.

A still relatively open issue is the licence that applies to external contributions. Generally it is assumed that someone who contributes to a project accepts *de facto* that their contribution adjusts to the conditions specified by its licence, although the legal grounds for this are poor. The initiative of the Free Software Foundation to ask by means of a (physical) letter for cession of all *copyright* from anyone who contributes more than ten lines of code to a GNU sub-project is a clear indication that in the world of free software there are stricter policies with regards to these contributions.

Based on the foregoing, in the rest of the chapter we will focus on analysing different licences. To place this analysis into context, we must remember that from now on, when we say that a licence is a free software licence, what we mean is that it fulfils the definitions of *free software* presented in section 1.1.1.

3.2.1. Types of licences

There is an enormous variety of free licences, although for practical reasons most projects use a small group of four or five. On the one hand, many projects don't want to or don't have the resources to design their own licence; on the other hand, most users prefer to refer to a well-known licence than having to read and analyse complete licences.

Bibliography

There is a compilation and discussion of the licences considered non-free or free but incompatible with the GPL from the point of view of the FSF in the Free Software Foundation, "Free licences" [121]. The philosophically different point of view of the Open Source Initiative is shown in its list (Open Source Initiative, "Open Source licences") [181]. We can see discrepancies in some licences, such as the Apple Public Source Licence Ver. 1.2, which the FSF considers non-free because of the obligation to publish all changes (even if they are private), to notify Apple of redistributions, or the possibility of unilateral revocation. Nevertheless, the pressure of this classification made Apple publish its version 2.0 in August 2003, which the FSF then did consider free.

It is possible to divide free software licences into two large families. The first comprises licences that do not impose special conditions on the *second redistribution* (in other words, that only specify that the software can be redistributed or modified, but that do not impose special conditions for doing so, which allows, for example, someone receiving the program to then redistribute it as proprietary software): these are what we will refer to as *permissive licences*. The second family, which we will call *strong licences* (or copyleft licences), include those that, in the style of GNU's GPL, impose conditions in the event of wanting to redistribute the software, aimed at ensuring compliance with the licence's conditions following the *first redistribution*. Whereas the first group emphasises the freedom of the person receiving the program to do almost anything they want with it (in terms of the conditions for future redistributions), the second emphasises the freedom of anyone who may potentially receive some day a work derived from the program, obliging subsequent modifications and redistributions to respect the terms of the original licence.

The difference between these two types of licences has been (and remains) a debatable issue amongst the free software community. In any case, we should remember that they are all free licences.

3.2.2.　Permissive licences

Permissive licences, also known sometimes as *liberal* or *minimal* licences, do not impose virtually any conditions on the person receiving the software, and yet, grant permission to use, redistribute and modify. From a certain point of view, this approach can be seen as a guarantee of maximum freedom for the person receiving the program. But from another, it may also be understood as maximum neglect in respect of ensuring that once someone receives the program, that person guarantees the same freedoms when redistributing that program. In practice, these licences typically allow software that its author distributes under a permissive licence to be redistributed with a proprietary licence.

Among these licences, the BSD licence is the best known, to such an extent that often permissive licences are referred to as *BSD-type* licences. The BSD (Berkeley Software Distribution) licence stems from the publication of different versions of Unix produced by the University of California in Berkeley, in the US. The only obligation it imposes is to credit the authors, while it allows redistribution in both binary and source code formats, without enforcing either of the two in any case. It also gives permission to make any changes and to be integrated into other programs without almost any restrictions.

Note

One of the consequences in practice of BSD-type licences has been to diffuse standards, since developers find no obstacle to making programs compatible with a reference implementation under this type of licence. In fact, this is one of the reasons for the extraordinary and rapid diffusion of Internet protocols and the *sockets*-based programming interface, because most commercial developers derived their implementation from the Berkeley University one.

Permissive licences are fairly popular, and there is an entire family with similar characteristics to the BSD: X Window, Tcl/Tk, Apache, etc. Historically, these licences appeared because the corresponding software was developed by universities with research projects financed by the US Government. The universities did without selling these programs, on the assumption that they had already been paid for by the Government, and therefore by the taxpayer, which meant that any company or individual could use the software without almost any restriction.

As already mentioned, on the basis of a program distributed under a permissive licence another one can be created (in reality, a new version), which may be proprietary. Critics of BSD licences see a danger in this feature, because it does not guarantee the freedom of future program versions. Its supporters, on the contrary, considered the ultimate expression of freedom and argue that, ultimately, you can do (almost) everything you want with the software.

Most permissive licences are a word for word copy of Berkeley's original, modifying just what refers to authorship. In some cases, such as the Apache project licence, it includes an additional clause, such as prohibiting giving the same name as the original to redistributed versions. All of these licences usually include, like BSD, the prohibition to use the name of the rightholder for promoting derived products.

At the same time, all the licences, whether BSD-type or not, include a *limitation of guarantee* which is really a *disclaimer*, necessary in order to avoid legal claims for implicit guarantees. Although this disclaimer in free software has been broadly criticised, it is common practice with proprietary software, which generally only guarantees that the medium is correct and that the program in question runs.

Summary outline of the BSD licence

Copyright © *the owner*. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1) Redistributions of source code must reproduce the *copyright* notice, and list these conditions and the disclaimer.

2) Redistributions in binary form must reproduce the *copyright* notice and list these conditions and the disclaimer in the documentation.

3) Neither the name of the *owner* nor the names of its contributors may be used to endorse products derived from this software without permission.

This program is provided "as is", and any express or implicit warranties, of merchantability or fitness for a particular purpose are disclaimed. In no event shall *the owner* be liable for any damage caused by its use (including loss of data, loss of profits or business interruption).

Next we describe in brief a few permissive licences:

- X Window licence, version 11 (X11)
  (http://www.x.org/Downloads_terms.html) [73].
  This is the licence used to distribute the X Window system, the most extensively used windows system in the world of Unix, and also GNU/Linux environments. It is very similar to the BSD licence, which allows redistribution, use and modification without practically any restrictions. It is sometimes called the *MIT licence* (with a dangerous lack of precision, since MIT has used other types of licences). Works derived from X Windows, such as XFree86 are also distributed under this licence.

- Zope Public Licence 2.0 (http://www.zope.org/Resources/ZPL) [76].
  This licence (commonly referred to as ZPL) is used for the distribution of Zope (an application server) and other related products. It is similar to BSD, with the curious feature that it expressly prohibits the use of trademarks registered by the Zope Corporation.

- Apache licence.
  This the licence under which most of the programs produced by the Apache project are distributed. It is similar to the BSD licence.

There are some free programs that are not distributed with a specific licence, rather the author explicitly declares them to be *public domain*. The main outcome of this declaration is that the author waives all rights to the program, which can therefore be modified, redistributed, used, etc., in any way. In practical terms, it is very similar to a program being under a BSD-type licence.

3.2.3. Strong licences

The GNU General Public Licence (GNU GPL)

The General Public Licence of the GNU project (Free Software Foundation, 1991) [118] (better known by its English acronym GPL), which appears in appendix C, is by far the most popular and well-known of all in the world of free software. It was created by the Free Software Foundation (promoter of the GNU project), and was originally designed to be the licence for all software generated by the FSF. However, its use has extended further becoming the most used licence (for example, more than 70% of the projects announced on Freshmeat are licensed under the GPL), even by flagship projects in the world of free software, such as the Linux kernel.

The GPL licence is interesting from a legal point of view because it makes creative use of *copyright* legislation, to achieve practically the opposite effect of what that legislation intends: instead of limiting users rights, it guarantees them. For this reason, this manoeuvre is frequently called *copyleft* (a play on words by replacing the word "right" with "left"). Someone with a sense of humour even devised the slogan "copyleft, all rights reversed".

In basic terms, the GPL licence allows redistribution in binary form and in source code, although in the case of a binary redistribution access to the source code is also obligatory. It also allows modifications to be made without any restrictions. However, it is only possible to redistribute code licensed under the GPL integrated with other code (for example, using *links*) if it has a compatible licence. This has been called the *viral effect* (although many consider this name to be disrespectful) of the GPL, since once code has been published with these conditions they can never be changed.

> **Note**
>
> A licence is incompatible with the GPL when it restricts any of the rights guaranteed by the GPL, either explicitly by contradicting any of its clauses, or implicitly, by imposing a new limitation. For example, the current BSD licence is compatible, but the Apache licence is incompatible because it demands that publicity materials expressly mention that combined work contains code of each and every one of the right holders. This does not imply that programs with both licences cannot be used simultaneously, or even be integrated. It just means that those integrated programs cannot be distributed, since it is impossible to comply simultaneously with the redistribution conditions of both.

The GPL licence is designed to guarantee the freedom of source code at all times, since a program published and licensed under its conditions may never be proprietary. Moreover, neither that program nor modifications of it may be published with a different licence other than the GPL. As already mentioned, supporters of the BSD-type licences see a limitation of freedom in that clause, whereas followers of the GPL believe that it is a way of ensuring that the software will always be free. One way of looking at it would be to consider that the GPL licence maximises the freedom of users, whereas the BSD licence maximises the freedom of developers. However, we should note that in the second case we are referring to developers in general and not to authors, since many

authors consider the GPL licence to be more in their interest, since it obliges competitors to publish their modifications (improvements, corrections, etc.) if they redistribute their software, whereas with a BSD-type licence this is not necessarily the case.

As regards this licence's contrary nature to *copyright*, it is because its philosophy (and that of the Free Software Foundation) is that software should not have owners (Richard Stallman, "Why software should not have owners", 1998) [207]. Although it is true that software licensed under the GPL has an author, who is the person that allows *copyright* legislation to apply to it, the conditions with which it is published confer such a nature on the software that we can consider ownership to correspond to the person in possession of the software and not to the person who has created it.

Of course, this licence also includes *disclaimers* in order to protect the authors. Likewise, in order to protect the good reputation of the original authors, any modification of a source file must include a note specifying the date and author of the modification.

The GPL also takes software patents into account, and demands that if the source carries patented algorithms (as we have said, something common and legal in the US, but currently irregular in Europe), either a licence for use of the patent free of charge must be granted, or it cannot be distributed under the GPL.

The latest version of the GPL licence, the second one, was published in 1991 (although at the time of writing the third one is in an advanced stage of preparation). Specifically bearing in mind future versions, the licence recommends licensing under the conditions of the second one or any other subsequent one published by the Free Software Foundation, which is what many authors do. Others, however, including in particular Linus Torvalds (Linux creator), only publish their software under the conditions of the second version of the GPL, in a bid to distance himself from potential future evolutions of the Free Software Foundation.

The third version of the GPL (http://gplv3.fsf.org) [115] intends to update it to the current software scenario, in respect mainly of aspects such as patents, DRM (*digital rights management*) and other limitations on software freedom. For example, the draft currently available at the time of writing (May 2007) does not allow a hardware manufacturer to block the use of certain software modules if they do not carry a digital signature accrediting a determined author. An example of this practice occurs with digital video recorders TiVo, which provide the source code to all their software (licenced with GPLv2) at the same time as they do not allow modifications of the code to be used on the hardware[4].

[4]This case has even suggested the use of the word *tivoisation* for other similar cases that have occurred.

Neither does the licence allow the software to be forcibly run on a pre-set environment, such as occurs when the use of unsigned kernels is prohibited on distributions that consider it appropriate for security reasons.

Note

There are several points in the GPLv3 licence that have generated a degree of opposition. One of the opposing groups is the group of Linux kernel developers (including Linus Torvalds himself). They consider that the requirement to use signed software components allows certain security features to be granted that would otherwise be impossible, at the same time as their explicit prohibition would extend the licence to the hardware field. Plus, the limitation established by the signatures mechanism would only occur with the hardware and software platforms that are designed that way, meaning that the software could be modified for its use on different hardware. In respect of this point, the FSF is in favour of using signature mechanisms that advise against using unsigned components for security reasons, but believes that not prohibiting those signatures mechanisms that prevent the use of unsigned components, could give rise to scenarios where there would be no hardware or software platforms on which to run those software modifications, meaning that the liberty of free software would then become totally limited where modifying code is concerned.

The GNU Lesser General Public Licence (GNU LGPL)

The GNU Lesser General Public Licence, (Free Software Foundation, GNU LGPL, version 2.1, February 1999) [119], commonly referred to by its initials in English - LGPL - is another licence of the Free Software Foundation. Designed initially for its use with libraries (the L, originally stood for *library*), it was recently modified to be considered the little sister (*lesser*) of the GPL.

The LGPL allows free programs to be used with proprietary software. The program itself is redistributed as if it were under the GPL licence, but its integration with any other software package is allowed without virtually any restrictions.

As we can see, originally this licence was aimed at libraries, in order to promote their use and development without encountering the integration problems implied by the GPL. However, when it was realised that the pursued objective of making free libraries popular was not compensated by the generation of free programs, the Free Software Foundation decided to change the *library* to *lesser* and advised against its use, except in very occasional and particular circumstances. Nowadays, there are many programs that are not libraries licensed under the terms of the LGPL. For example, the Mozilla navigator or OpenOffice.org office *suite* are also licensed, among others, under the LGPL.

Note

As is the case with the GPL, the last published version of the LGPL is the second, although there is already a template of the third version (http://gplv3.fsf.org/pipermail/info-gplv3/2006-July/000008.html) [116]. This new version is shorter than the previous one since it refers all its text to the GPLv3 and merely highlights its differences.

Other strong licences

Other strong licences that deserve mentioning:

- Sleepycat license (www.sleepycat.com/download/oslicense.html) [59].
  This is the license under which the company Sleepycat (http://www.sleepycat.com/) [60] distributes its programs (of which we could mention the well-known Berkeley DB). It enforces certain conditions whenever the program or works derived from it are redistributed. In particular, it obliges the source code to be offered (including the modifications in the case of a derived work) and for the redistribution to impose the same conditions on the receiver. Although much shorter than the GNU GPL, it is very similar in its main effects.

- eCos License 2.0 (http://www.gnu.org/licenses/ecos-license.html) [25].
  This is the license under which eCos (http://sources.redhat.com/ecos/) [24], a real-time operating system, is distributed. It is a modification of the GNU GPL which does not consider that code linked to the programs it protects, is subject to the clauses of the GNU GPL if redistributed. From this point of view, its effects are similar to those of the GNU LGPL.

- Affero General Public License (http://www.affero.org/oagpl.html) [78].
  It is an interesting modification of the GNU GPL which considers the case of programs offering services via the web, or in general, via computer networks. This type of program represents a problem from the point of view of strong licences. Since use of the program does not imply having received it through a redistribution, even though it is licensed, under the GNU GPL for example, someone can modify it and offer a service on the Web using it, without redistributing it in any way, and therefore, without being obliged, for example, to distribute its source code. The Affero GPL has a clause obliging that if the program has a means for providing its source code via the web to whoever uses it; this feature may not be disabled. This means that if the original author includes this capability in the source code, any user can obtain it, and plus that *redistribution* is subject to the conditions of the licence. The Free Software Foundation is considering including similar provisions in version 3 of its GNU GPL.

- IBM Public License 1.0 (http://oss.software.ibm.com/developer-works/opensource/license10.html) [40].
  It is a licence that allows a binary redistribution of derived works only if (among other conditions) a mechanism is contemplated for the person receiving the program to receive the source code. The redistribution of source code must be made under the same licence. This licence is also interesting because it obliges the party redistributing the program with modifications, to license automatically and free of charge any patents affecting such modifications and that are the property of the redistributor to the party receiving the program.

- Mozilla Public License 1.1 (http://www.mozilla.org/MPL/MPL-1.1.html) [49].

This is an example of a free licence drawn up by a company. It is an evolution of the first free licence that Netscape Navigator had, which was very important in its day because it was the first time that a well-known company decided to distribute a program under its own free licence.

3.2.4.   Distribution under several licences

Up until now we have assumed that every program is distributed under a single licence which specifies the conditions for use and redistribution. However, an author can distribute works under different licences. In order to understand this, we should remember that every publication is a new work, and that different versions can be distributed with the only difference being in their licence. As we will see, most of the time this translates into the fact that depending on what the user wants to do with the software he will have to observe the terms of one licence or another.

One of the best known examples of a double licence is the one for the Qt library, on which the KDE desktop environment is founded. Trolltech, a company based in Norway, distributed Qt with a proprietary licence, although it waived payment for programs that didn't use it for profit. For this reason and because of its technical characteristics, it was the KDE project's choice in the mid-nineties. This gave rise to an intense controversy with the Free Software Foundation because then KDE stopped being completely free software, as it depended on a proprietary library. Following an extensive debate (during which GNOME appeared as KDE's free competitor in the desktop environment), Trolltech decided to use the double-licence system for its star product: the programs under the GPL could use a Qt GPL version, whereas if the intention was to integrate it with programs that had incompatible licences with the GPL (such as proprietary licences), a special licence had to be bought from them. This solution satisfied all parties, and nowadays KDE is considered free software.

Other well-known examples of dual licences are StarOffice and OpenOffice.org, or Netscape Communicator and Mozilla. In both cases, the first product is proprietary whereas the second is a free version (generally under the conditions of several free licences). Although originally free projects were limited versions of their proprietary siblings, over time they have followed their own path, meaning that nowadays they have a fairly high level of independence.

3.2.5.   Program documentation

The documentation that comes with a program forms an integral part of it, as do the comments on source code, as recognised, for example by the Spanish Law on Intellectual Property. Given this level of integration, it would seem

logical that the same freedoms should apply to the documentation and that it should evolve in the same way as the program: any modification made in a program requires a simultaneous and consistent change in its documentation.

Most of this documentation tends to be coded as unformatted text files, since the aim is to make it universally accessible with a minimum tools environment, and (in the case of free programs) normally includes a small introduction to the program (README), installation guidelines (INSTALL), some history on the evolution and future of the program (CHANGELOG and TODO), authors and copyright (AUTHORS and COPYRIGHT or COPYING), as well as the instructions for use. All of these, excluding authors and copyright, must be freely modifiable as the program evolves. To authors we just need to add names and credits without eliminating anything, and the copyright must only be modified if the conditions allow it.

The instructions for use are normally coded in more complex formats, since they tend to be longer and richer documents. Free software demands that this documentation may be changed easily, which in turn enforces the use of so-called *transparent* formats, with known specifications and able to be processed by free tools, such as, in addition to pure and clean text, the format of the Unix manual pages, TexInfo, LaTeX or DocBook, without prejudice to also being able to distribute the result of transforming these source documents into more suitable formats for visualisation or printing, such as HTML, PDF or RTF (normally more *opaque* formats).

However, program documentation is often prepared by third parties who have not been involved in the development. Sometimes the documentation is of a didactic nature, to facilitate the installation and use of a specific program (HOWTO); sometimes it is more extensive documentation that covers several programs and their integration, that compares solutions, etc., either in the form of a tutorial or reference manual; sometimes it is a mere compilation of frequently asked questions and their answers (FAQ). A noteworthy example is the Linux documentation project (http://www.tldp.org) [44]. In this category we could also include other technical documents, not necessarily about programs, whether the instructions for cabling a local network, making a solar oven, repairing an engine or selecting a tools supplier.

These documents are halfway between mere program documentation and highly technical and practical articles or books. Without prejudice to the freedom to read, copy, modify and redistribute, the author may wish to give opinions that he does not want to be distorted, or at least not want any distortion to be attributed to him; or he may wish to keep paragraphs, such as acknowledgements; or make it forcible to modify others, such as the title. Although these concerns can also arise with software itself, they have not been expressed as vehemently in the world of free software as in the world of free documentation.

3.3. Summary

In this chapter we have looked at the legal aspects that govern or influence the world of free software. They form part of intellectual or industrial property legislation conceived, in principle, to stimulate creativity by rewarding creators for a specific period. Of the different types, so-called *copyright* is the one that most affects free software, and properly applied it can be used to guarantee the existence of free software in the form of free licences.

We have seen the importance of licences in the world of free software. And we have also presented the enormous variety of existing licences, the grounds on which they are based, their repercussions, advantages and disadvantages. Certainly, we can say that the GPL tries to maximise the freedom of software users, whether they receive the free software directly from its author or not, whereas BSD-type licences try to maximise the freedom of the modifier or redistributor.

Given what we have seen in this chapter, we can deduce that it is very important to decide early on what licence a project will have and to be fully aware of its advantages and disadvantages, since a later modification tends to be extremely difficult, especially if the number of external contributions is very large.

To conclude, we would like to highlight the fact that free software and proprietary software differ solely and exclusively in terms of the licence under which the programs are published. In the following chapters, however, we will see that this purely legal difference may or may not affect the way software is developed, giving rise to a new development model, which can differ from the "traditional" development methods used in the software industry to a greater or lesser extent, depending on each case.

4. Developers and their motivations

"Being a *hacker* is lots of fun, but it's a kind of fun that takes a lot of effort. The effort takes motivation. Successful athletes get their motivation from a kind of physical delight in making their bodies perform, in pushing themselves past their own physical limits. Similarly, to be a *hacker* you have to get a basic thrill from solving problems, sharpening your skills and exercising your intelligence."

Eric Steven Raymond, "How to become a hacker"

4.1. Introduction

The partly anonymous and distributed way in which free software is developed has meant that for many years the human resources that it relies on have been largely unknown. The result of this lack of knowledge has been to mythologise, at least to some extent, the world of free software and the life of those behind it, based on more or less broad stereotypes about the *hacker* culture and computers. In the last few years, the scientific community has made an enormous effort to get to know the people who participate in free software projects better, their motivations, academic backgrounds, and other potentially relevant aspects. From a purely pragmatic point of view, knowing who is involved in this type of projects and why, can be extremely useful when it comes to generating free software. Some scientists, mainly economists, psychologists, and sociologists, have wanted to go further and have seen in this community the seed of future virtual communities with their own rules and hierarchies, in many cases totally different to those we know in "traditional" society. One of the most important mysteries to resolve was to learn what motivated software developers to participate in a community of this nature, given the fact that financial benefits, at least direct ones, are practically nonexistent, whereas indirect ones are difficult to quantify.

4.2. Who are developers?

This section aims to provide a global overview of the people who spend their time and energy participating in free software projects. The data that we show stems mostly from scientific research in the last few years, the most significant but by no means exclusive including *Free/libre and open source software. Survey and study*, part IV: "Survey of developers", 2002 [126], and "Who is doing it? Knowing more about libre software developers", 2001 [197].

Software developers are normally young people. The average age is around twenty seven. The variation in age is significant, since the dominant group is in the twenty one to twenty four age bracket, and the most frequently appearing value is twenty three. It is interesting to note how the age of joining the free software movement peaks between eighteen and twenty five and is particularly pronounced between twenty one and twenty three, which would

coincide with university age. This evidence stands in contrast to the claim that free software is mostly a teenage thing, although there is an obvious involvement of teenagers (about 20% of developers are under twenty). For certain, what we can see is that most developers (60%) are in their twenties, with the under-twenties and over-thirties equally sharing the remaining 40%.

From the age of joining we can deduce that there is an enormous university influence on free software. This is not surprising, given that as we have seen in the chapter on history, before free software was even known by that name it was closely connected to higher education. Even today, student user groups and universities continue to drive the use and expansion of free software. Therefore, it is not surprising that more than 70% of developers have a university education. This data is even more significant if we bear in mind that the remaining 30% are not at university yet because they are still in school. Even so, they are also involved and are no less appreciated than developers who have never had access to higher education, but are IT enthusiasts.

The free software developer is normally male. The figures juggled by different surveys on the presence of women in the community vary between 1% and 3%, competing with their own margin of error. At the same time, a majority (60%) claims to have a partner, while the number of developers with children is just 16%. Given the age brackets of free software developers, this data coincides fairly accurately with a random sample, meaning that it may be considered "normal". The myth of the lonely developer whose enthusiasm for IT is the only thing in his life is shown to be, as we can see, an exception rather than the rule.

4.3. What do developers do?

Professionally speaking, free software developers describe themselves as software engineers (33%), students (21%), programmers (11%), consultants (10%), university professors (7%), etc. On the opposite end of the scale, we find that they tend not to form part of sales or marketing departments (about 1%). It is interesting to note how many of them define themselves as software engineers rather than programmers, almost three times as many, bearing in mind, as we will see in the chapter on software engineering, that the application of classical software engineering techniques (and even some modern ones) is not really entrenched in the world of free software.

The university connection, which has already been proven, rears its head again in this section. About one in three developers is a student or university professor, which goes to show the tremendous collaboration between people mainly from the software industry (the remaining two thirds) and the academic sphere.

At the same time, it has been possible to identify a large scope of mixed disciplines: one in five developers comes from a field that is not IT. This, combined with the fact that there is also a similar number of non-university developers, reflects a wealth of interests, origins, and certainly, composition of development teams. It is very difficult to find a modern industry, if there is one, where the degree of heterogeneity is as large as the one we can see in free software.

In addition to the approximately 20% of students, 64% of developers are mostly paid employees, whereas the proportion of self-employed developers is 14%. Finally, just 3% claims to be unemployed, a significant fact since the survey was conducted after the dotcom crisis began.

Note

The fact that free software business model, unlike with proprietary software, cannot be achieved through the sale of licences has always propitiated heated debates as to how programmers should earn their living. In the surveys that we refer to in this chapter, 50% of developers claimed to have obtained direct or indirect financial compensation for their involvement in free software. However, many others aren't so sure. Richard Stallman, founder of the GNU project, when asked what a free software developer should do in order to make money, tends to reply that he can work as a waiter.

4.4. Geographical distribution

Obtaining developers' geographical data is an issue that needs to be approached in a more scientific manner. The problem with the research shown in this chapter is that because it is based on Internet surveys, open to anyone wishing to participate, participation has depended to a great extent on the sites it was posted, and the way in which it was announced. To be accurate, we should note that the surveys did not aim to be representative in this regard, but rather to obtain the answers and/or opinions of the largest possible number of free software developers.

However, we could venture to make a few statements on this issue, knowing that this data is not as reliable as previous data, and that therefore, the margin of error is much greater. What seems to be a fact is that most free software developers come from industrialised countries, and that the presence of developers from so-called Third World countries is rare. Consequently, it shouldn't be surprising that the map of developers of the Debian project (http://www.debian.org/devel/developers.loc) [187], for example, matches the photographs of the Earth at night: where there is light - read "where there is an industrialised civilisation" - that is where free software developers tend to concentrate. This, which may seem logical at first sight, stands in contrast to the potential opportunities that free software can offer Third World countries.

We can find a clear example in the following table, which contains the most common countries of origin of the Debian project developers in the last four years. There is a noticeable trend towards decentralisation of the project, evident from the fact that the growth in the number of developers from the US, the country which most contributes, is lower than the average. And the

fact is that, in general, countries have managed to double their numbers of volunteers in the last four years, and France, which has managed to multiply its presence by five, is the clearest example in this regard. Considering that Debian took its first steps on the American continent (in the US and in Canada in particular), we can see that in the last few years the project has become *Europeanised*. We assume that the following step will be the much sought-after globalisation, with the incorporation of South American, African and Asian countries (with the exception of Korea and Japan, which are already well represented), although the data in our possession (two collaborators from Egypt, China or India, and one in Mexico, Turkey or Colombia in June 2003) are not very promising in this sense.

In the world of free software (and not just in the case of Debian), there is an extensive debate over the supremacy of Europe or the United States. Almost all studies have shown that the presence of European developers is slightly higher than the North American one, an effect that is mitigated by the fact that Europe's population is greater than that of the US. Therefore, we are dealing with a war of numbers, since the number of developers per capita is higher among the North Americans. If we take into account the number of people with Internet access instead of the total population, then Europe comes out on top again.

In terms of countries, the areas with the highest levels of implantation (in numbers of developers divided by the population) are Northern Europe (Finland, Sweden, Norway, Denmark and Iceland) and Central Europe (Benelux, Germany and the Czech Republic), followed by Australia, Canada, New Zealand and the US. Despite its importance in absolute terms (due to the large populations of France, Italy and Spain), the Mediterranean zone however, is below average.

Table 1. Countries with the largest number of Debian developers

Country	01/07/1999	01/07/2000	01/07/2001	01/07/2002	20/06/2003
US	162	169	256	278	297
Germany	54	58	101	121	136
UK	34	34	55	63	75
Australia	23	26	41	49	52
France	11	11	24	44	51
Canada	20	22	41	47	49
Spain	10	11	25	31	34
Japan	15	15	27	33	33
Italy	9	9	22	26	31
Netherlands	14	14	27	29	29

Country	01/07/1999	01/07/2000	01/07/2001	01/07/2002	20/06/2003
Sweden	13	13	20	24	27

4.5. Dedication

The number of hours that free software developers spend on developing free software is one of the big unknowns. We should also point out that this is one of the main differences with company-generated software, where the members of the team and the time spent by each team member on a development are well known. The time that developers dedicate to free software can be taken as an indirect measure of their level of professionalisation. Before showing the data currently available, it is important to note that it has been obtained from estimates given by developers in various surveys, so that in addition to the inherent inaccuracies of this type of data gathering, we need to consider a margin of error associated to how each developer interprets development time. Hence, it is certain that many developers will not include the time spent reading e-mails (or perhaps they will) and only specify the time they spend programming and debugging. Therefore, all the figures we show next need to be considered with due reserve.

The research conducted to date shows that each software developer spends eleven hours a week on average ("Motivation of software developers in open source projects: an internet-based survey of contributors to the Linux kernel", 2003) [143]. However, this figure can be deceptive, since there is an enormous variation in the time spent by software developers. In the study *Free/libre and open source software. Survey and study*, part IV: "Survey of developers", 2002 [126], 22.5% of those surveyed said that their contribution was less than two hours per week, and this figure increased to 26.5% for those spending two to five hours per week; between six and ten hours was the time spent by 21.0%, while 14.1% spent between eleven and twenty hours per week; 9.2% claimed that the time they spent developing free software was between twenty and forty hours per week and 7.1%, over forty hours per week.

Table 2. Dedication in hours per week

Hours per week	Percentage
Less than 2 hours	22.5%
Between 2 and 5 hours	26.1%
Between 5 and 10 hours	21.0%
Between 10 and 20 hours	14.1%
Between 20 and 40 hours	9.2%
More than 40 hours	7.1%

Note

In addition to showing the level of professionalisation of free software development teams, the time spent in hours is a relevant parameter when it comes to making cost estimates and comparisons with proprietary development models in the industry. With free software, for now, we just have the end products (new software deliveries, synchronisation of new code in versions systems...) which do not allow us to know how much time the developer has spent on achieving them.

An analysis of these figures tell us that about 80% of developers perform these tasks in their free time, whereas only one in five could consider that they spend as much time on this activity as a professional. Later, in the chapter on software engineering, we will see how this data matches developers' contributions, since they both appear to follow the Pareto law (*vid.* section 7.6).

4.6. Motivations

There has been much speculation as to the motivations that developers have to develop free software, especially when it is done in free time (which, as we have seen, corresponds to about 80% of developers). As in previous sections, we only have the survey data, which is why it is important to realise that the answers have been given by the developers themselves, meaning that they may be more or less coherent with reality. The percentages shown next exceed the 100% mark because there was an option for participants to select several answers.

In any case, it appears from their answers that most want to learn and to develop new skills (approximately 80%) and that many do so in order to share knowledge and skills (50%) or to participate in a new form of cooperation (about a third). The first data is not surprising, given that a professional with more knowledge will be in greater demand than one with less. However, it is not quite so easy to explain the second data, and it would even seem to contradict Nikolai Bezroukov's opinion in "A second look at the cathedral and the bazaar" (December, 1998) [91] that the leaders of free software projects are careful not to share all the information in their possession in order to perpetuate their power. Meanwhile, the third most frequent choice is undoubtedly, a true reflection of developers' enthusiasm about the way free software is created in general; it is difficult to find an industry in which a group of lightly organised volunteers can -technologically speaking- stand up to the sector's giants.

Although the "classical" theory for explaining why free software developers spend time contributing to this type of projects is reputation and indirect financial benefits in the medium and long term, it would appear that developers themselves disagree with these claims. Just 5% of those surveyed replied that they develop free software in order to make money, whereas the number who did so in order to establish a reputation rose to 9%, far from the answers given in the preceding paragraph. In any case, it seems that researching developers' motivations to become part of the free software community is a fundamental task, which sociologists and psychologists will have to face in the near future.

4.7. Leadership

Reputation and leadership are two of the characteristics used to explain the success of free software, and especially, the bazaar model, as we will see from the chapter on software engineering. As we have seen in another chapter, on software licences, there are certain differences between free software licences and its equivalents in the documentation field. These differences stem from the way of retaining authorship and authors' more accentuated opinion in text than in programs.

In *Free/libre and open source software. Survey and study*, part IV: "Survey of developers" (2002) [126] a question was included that asked developers to point out what people from a list were known to them, not necessarily personally. The results, set out in table 3, show that these people can be divided into three clearly distinct groups:

Table 3. Level of awareness of important developers

Developer	Known for
Linus Torvalds	96.5%
Richard Stallman	93.3%
Miguel de Icaza	82.1%
Eric Raymond	81.1%
Bruce Perens	57.7%
Jamie Zawinski	35.8%
Mathias Ettrich	34.2%
Jörg Schilling	21.5%
Marco Pesenti Gritti	5.7%
Bryan Andrews	5.6%
Guenter Bartsch	3.5%
Arpad Gereoffy	3.3%
Martin Hoffstede	2.9%
Angelo Roulini	2.6%
Sal Valliger	1.2%

- A first group of people with clear philosophical and historical connotations within the world of free software (although, as we know, they may also have notable technical skills):

1) Linus Torvalds. Creator of the Linux kernel, the most used free operating system, and co-author of *Just for fun: the story of an accidental revolutionary* [217].

2) Richard Stallman. Ideologist and founder of the Free Software Foundation and developer on various GNU projects. Author of several important essays on free software ("Why *free software* is better than *open source*", 1998 [206], "Copyleft: pragmatic idealism", 1998 [205], "The GNU Project" [208] and "The GNU Manifesto", 1985 [117]).

3) Miguel de Icaza. Co-founder of the GNOME project and Ximian Inc., and developer in GNOME and MONO.

4) Eric Raymond. Promoter of the Open Source Initiative, author of "The cathedral and the bazaar" [192] and main developer of fetchmail.

5) Bruce Perens. Former leader of the Debian project, promoter (converted) of the Open Source Initiative and developer of the e-fence tool.

6) Jamie Zawinsky. Ex developer of Mozilla and famous for a letter of 1999 in which he left the Mozilla project arguing that the model they were following would never bear fruit ("Resignation and postmortem", 1999) [237].

7) Mathias Ettrich. Founder of KDE and developer of LyX and others.

• A second group consisting of developers. This survey took the names of the leading developers of the six most popular projects according to the FreshMeat free software download site. We can see that (with the exception of Linus Torvalds, for obvious reasons, and Jörg Schilling) the level of awareness of these developers is small:

1) Jörg Schilling, creator of cdrecord, among other applications.

2) Marco Pesenti Gritti, main developer of Galeon.

3) Bryan Andrews, developer of Apache Toolbox.

4) Guenther Bartsch, creator of Xine.

5) Arpad Gereoffy, developer of MPEGPlayer.

• A third group consisting of the names of the three last "people" in the table. These names were invented by the survey team in order to check the margin of error.

We can draw two conclusions from the results: the first is that we can consider the margin of error to be small (less than 3%), and the second is that most developers of the most popular free software applications are as well-known as people who do not exist. This data should make those who allege that one of the main reasons for developing free software is fame-seeking think twice.

4.8. Summary and conclusions

This chapter has attempted to shed some light on the largely unknown issue of the people who dedicate time to free software. In general terms, we can say that a free software developer is a young male with a university qualification (or on the way to getting one). The relationship between the world of free software and universities (students and professors) is very close, although the developer who has nothing to do with the academic sphere still predominates.

In terms of hours of dedication, we have shown that there is an enormous disparity, similar to what is postulated in the Pareto law. Developers' motivations, in their own opinion, are far from being monetary and egocentric, as economists and psychologists tend to suggest, and are mostly to do with sharing and learning. Finally, we have shown a table of the most significant personalities in the world of free software (including others who were not so well-known, as we have seen) and shown that reputation in the enormous free software community tends to depend on more than just coding a successful free software application.

5. Economy

"Res publica non dominetur."

"Public things have no owner." (free translation)

Appeared in an IBM advert about Linux (2003)

This chapter looks at some economic aspects related to free software. We will start by showing how free software projects are financed (when they are indeed financed, since in many cases they rely solely on efforts and resources contributed voluntarily). Next, we will look at the main business models put into practice by companies directly associated to free software. The chapter ends with a small study of the relationship between free software and monopolies in the software industry.

5.1. Funding free software projects

Free software is developed in many different ways and using mechanisms to obtain funds that vary enormously from case to case. Every free project has its own way of financing itself, from the one consisting totally of volunteer developers and using only altruistically ceded funds, to the one carried out by a company that invoices 100% of its costs to an organisation interested in the corresponding development.

In this section, we will focus on the projects where there is external funding and not all the work is voluntary. In these cases, there is normally some form of cash inflow, from an external source to the project, responsible for providing funds for its development. This way, the free software that is built may be considered, to some extent, to be the product of this external funding. This is why it is common for that external source to decide (at least in part) how the funds are spent and on what.

In some way, this external funding for free projects can be considered a kind of sponsorship, although this sponsorship has no reason for being disinterested (and usually it is not). In the following sections we discuss the most common types of external funding. However, while learning about them, we should remember that these are just some of the ways that free software projects obtain resources. But there are others, and of these the most important one (as we have seen in chapter 4) is the work of many volunteer developers.

5.1.1. Public funding

A very special type of financing for free projects is public funding. The funding body may be directly a government (local, regional, national or even supranational) or a public institution (for example, a foundation). In these cases, the

funding tends to be similar as for research and development projects, and in fact it is common for the funding to come from public bodies that promote R +D. Normally, the funding body will not seek to recover the investment (or at least not directly), although it tends to have clear objectives (such as promoting the creation of an industrial or research-based fabric, promoting a certain technology or a certain type of application, etc.).

In most of these cases, there is no explicit financing of products or services related to free software, but rather this tends to be the sub-product of a contract with other more general objectives. For example, as part of its research programs, the European Commission funds projects aimed at improving European competitiveness in certain fields. Some of these projects have as part of their objectives to use, improve and create free software within the scope of the research (as a research tool or a product derived from it).

The motivations for this type of financing are very varied, but we can distinguish the following:

1) Scientific. This is the most frequent one in the case of publicly funded research projects. Although its objective is not to produce software but rather to investigate a specific field (whether IT-related or not), it is likely to require programs to be developed as tools for achieving the project's objectives. Usually the project is not interested in commercialising these tools, or may even be actively interested in other groups using and improving them. In such cases, it is fairly common to distribute them as free software. In this way, the group conducting the research has partly dedicated funds to producing this software, so we can say that it has been developed with public funding.

2) Promoting standards. Having a reference implementation is one of the best ways of promoting a standard. In many cases this involves having programs that form part of said implementation (or if the standard refers to the software field, to be the implementation themselves). For the reference implementation to be useful in promoting the standard, it needs to be available, at least in order to check interoperativity for all those wishing to develop products that subscribe to that standard. And in many cases it is also advisable for manufacturers to be able to adapt the reference implementation directly in order to use it with their products if they wish. This is how, for example, the Internet protocols were developed, which have now become a universal norm. In such cases, releasing reference implementations as free software can contribute enormously towards that promotion. Once again, free software here is a sub-product, in the case of promoting a standard. And normally the party responsible for this promotion is a public body (although sometimes it may be a private consortium).

3) Social. Free software is a very interesting tool for creating the basic infrastructure for the information society. Organisations interested in using

free software to promote universal access to the information society can finance projects related to it (normally with projects for developing new applications or adapting already existing ones).

Note

An example of public financing for a primarily social objective is the case of LinEx, promed by the Extremadura Regional Government (Extremadura, Spain) in order to promote the information society fundamentally in terms of computer literacy. The Regional Government has financed the development of a distribution based on Debian in order to achieve this objective. Another similar case is the German government funding of GnuPG developments, aimed at making them easier to use for inexperienced users, with the idea of promoting the use of secure mail by its citizens.

The development of GNAT

A notorious case of public financing for a free software development is the case of the GNAT compiler. GNAT, the Ada compiler, was financed by the Ada 9X project of the US Department of Defence, with the idea of having a compiler of the new version of the Ada programming language (which would later become Ada 95), which it was trying to promote at that time. One of the causes identified in relation to software companies adopting Ada's first version (Ada 83) was the late availability of a language compiler and its high cost when it was finally released. Therefore, they tried to prevent the same thing from happening with Ada 95, ensuring that the compiler was ready almost simultaneously with the release of the language's new standard.

To do so, Ada 9X contracted a project with a team from the University of New York (NYU), for an approximate value of one million USD, to carry out a "concept implementation" of the Ada 95 compiler. Using these funds, and taking advantage of the existence of GCC (GNU's C compiler, of which most of the backend was used), the NYU team effectively built the first Ada 95 compiler, which it released under the GNU GPL. The compiler was so successful that when the project was finished some of its creators founded a company (Ada Core Technologies), which since then has become the market leader in compilers and help tools for building programs in Ada.

In this project it is worthy to note the combination of research elements (in fact, this project advanced knowledge on the building of front ends and run time systems for Ada-type language compilers) and promotion of standards (which was the funding body's clearest objective).

5.1.2. Private not-for-profit funding

This type of funding has many similar characteristics to the previous type, which is normally conducted by foundations or non-governmental organisations. Direct motivation in these cases tends to be to produce free software for use in a sphere that the funding body considers particularly relevant, but we may also find the indirect motivation of contributing to problem-solving (for example, a foundation that promotes research into a disease may finance the construction of a statistics program that helps to analyse the experimental groups used as part of the research into that disease).

In general, both the motives and the mechanisms for this type of funding are very similar to those of public funding, although naturally they are always in the context of the funding body's objectives.

Note

Probably, the archetypal case of a foundation that promotes the development of free software is the Free Software Foundation (FSF). Since the mid-1980s this foundation is dedicated to promoting the GNU project and to encouraging the development of free software in general.

Another interesting case, although in a rather separate field, is the Open Bioinformatics Foundation. The objectives of this foundation include promoting the development of basic computer programs for research in any of the branches of bioinformatics. An in general, it promotes this by financing and contributing to the construction of free programs.

5.1.3. Financing by someone requiring improvements

Another type of financing the development of free software, which is not quite so altruistic, takes place when someone needs to make improvements to a free product. For example, for internal use, a company may need a certain program to have a particular functionality or to correct a few bugs. In these cases, it is common for the company in question to contract the required development. This development is often free software (whether because the licence of the modified program imposes it, or because the company decides it).

The case of Corel and Wine

Towards the end of the 1990s, Corel decided to port its products to GNU/Linux. During this process it discovered that a free program designed to facilitate the execution of binaries for Windows in Linux environments could help to make considerable development savings. But in order to do so, it had to be improved, fundamentally by adding the emulation of some Windows functionality that the Corel programs used.

For this, Corel contracted Macadamian, which contributed its improvements to the Wine project. This way, both Corel and Wine benefited.

5.1.4. Funding with related benefits

With this type of financing, the funding body aims to obtain benefits from products related to the program whose development it funds. Normally, in these cases the benefits obtained by the funding body are not exclusive, since others can also enter the market for selling the related products, but either the market share it captures is sufficient for it not to worry too much about sharing the pie with others, or it has a clear competitive advantage.

Some examples of products related to a particular software are as follows:

*   Books. The company in question sells manuals, user guides, course materials, etc. related to the free program that it helps to finance. Of course, other companies can also sell related books, but normally financing the project will give the company early access to key developers before the competition, or simply provide a good image towards the user community of the program in question.

*   Hardware. If a company funds the development of free systems for a certain type of hardware, it can more easily dedicate itself to selling that type

of hardware. Once again, since the software developed is free, competitors selling the same type of devices may appear, that use the same developments without having collaborated in the funding. But even so, the company in question has several advantages over its competitors, and one of them may be that its position as a source of funding for the project allows it to exert influence so that priority is given to the developments in which it is most interested.

- CD with programs. Probably, the best known model of this type is the one of companies financing certain developments that they then apply to their software distribution. For example, having a good desktop environment can help a lot to sell a CD with a certain distribution of GNU/ Linux, and therefore, financing its development could be a good business for the party selling the CDs.

We need to bear in mind that under this heading the financing in question has to be made with a profit motivation, and therefore the funding body has to obtain a potential benefit from the financing. In reality, however, it is common for there to be a combination of profit motive and altruism when a company provides funds for a free program to be made from which it expects to benefit indirectly.

Note

A well-known case of funds contributed to a project, albeit fairly indirectly, is the help that the O'Reilly publishing house gives to the development of Perl. Naturally, it is no coincidence that O'Reilly is also one of the main publishers of subjects related to Perl. In any case, it is obvious that O'Reilly does not have exclusivity over the publication of books of this kind, and that other publishing houses compete in this market segment, with varying degrees of success.

VA Software (originally VA Research and later VA Linux) has collaborated actively in developing the Linux kernel. Through this, it has achieved, among others, ensured continuity, which was particularly critical for it in relation to its customers when its main business was selling equipment with a GNU/Linux pre-installation.

Red Hat has financed the development of many GNOME components, essentially obtaining a desktop environment for its distribution, which has contributed to increasing its sales. As in previous cases, other manufacturers of distributions have benefited from this development, although many of them have not collaborated with the GNOME project to the same extent as Red Hat (and quite a few have not collaborated at all). Despite this fact, Red Hat benefits from its contribution to GNOME.

5.1.5. Financing as an internal investment

There are companies that develop free software directly as part of their business model. For example, a company may decide to start a new free project in a field where it believes that there are business opportunities, with the idea of subsequently obtaining a return on that investment. This model could be considered a variation of the previous one (indirect financing), and the "related benefits" would be the advantages that the company obtains from producing the free program. But since in this case it is the free product itself which is expected to produce the benefits, it seems appropriate to give it its own heading.

This type of financing gives rise to various business models. When we analyse them (in section 5.2) we will also explain the advantages that a company normally obtains from this type of investment in a project and what methods tend to be used in order to make it profitable. But in any case, we should mention that sometimes the software in question may be developed simply in order to satisfy the company's own needs, and that only later the company may decide to release it, and perhaps, to open a business line based on it.

Note

Digital Creations (now Zope Corporation) is one of the most well-known cases of a company dedicated to developing free software with the expectation of making a return on its investment. The free project that Zope invests most heavily in is an applications server that is enjoying a certain amount of success. Its history with free software started when the then Digital Creations was looking for venture capital to develop its proprietary applications server, around 1998. One of the groups most interested in investing in them (Opticality Ventures) established as a condition that the resulting product must be free, because otherwise they did not see how they could obtain a significant market share. Digital Creations agreed to this approach and a few months later announced the first version of Zope (it changed its name a few years later). Nowadays, Zope Corporation is specialised in consulting, training and support for content management systems based on Zope, and other products of which Zope is unquestionably the cornerstone.

Ximian (formerly Helix Code) is a well-known case of free applications development in a business environment. Closely linked since its origins to the GNOME project, Ximian has produced software systems such as Evolution (a personal information manager which includes a relatively similar functionality to Microsoft Outlook), Red Carpet (an easy-to-use system for managing packages on an operating system) and MONO (an implementation of a large part of .NET). The company was founded in October 1999 and attracted many developers from GNOME, who became members of its development team (while continuing in many cases to collaborate with the GNOME project). Ximian positioned itself as an engineering company expert in GNOME adaptations, in building applications based on GNOME, and in general, in providing development services based on free software, especially tools related to the desktop environment. In August 2003, Ximian was bought by Novell.

Cisco Enterprise Print System (CEPS) (http://ceps.sourceforge.net/) [17] is a printing management system for organisations that use very many printers. It was developed internally in Cisco to satisfy its own needs and freed in 2000 under the GNU GPL. It is difficult to know for sure Cisco's reasons for doing this, but they could be related to finding external contributions (error reports, new controllers, patches, etc.). In any case, what is obvious is that since Cisco had no plans to commercialise the product and its potential market was not very clear, it had very little to lose with this decision.

5.1.6. Other financing modes

There are other financing modes that are difficult to classify under the previous headings. As an example, we could mention the following:

• Use of the market for putting developers and clients in contact. The idea that sustains this mode of financing is that, especially for minor developments, it is difficult for a client wanting a specific development to come into contact with a developer capable of doing it in an efficient manner. In order to improve this situation, free software development markets are established where developers can advertise their skills and clients, the developments that they need. A developer and a client reach an agreement; we have a similar situation to the one already described as "funding by a party requiring improvements " (section 5.1.3).

SourceXchange

SourceXchange is an example of a market that put developers in contact with potential clients. To advertise a project, a client would present an RFP (*request for proposal*) specifying the development required and the resources it was prepared to provide for that development. These RFPs were published on the site. When a developer read one that interested him, he would make an offer for it. If a developer and a client agreed on the terms of the development, a project would begin. Normally, every project was supervised by a *peer reviewer*, a reviewer responsible for ensuring that the developer complied with the specifications and that indeed the specifications made sense, and advising on how to carry through the project, etc. SourceXchange (owned by the company CollabNet) took charge of providing the site, guaranteeing reviewers' capabilities, ensuring payment in the case of completed projects and offering monitoring tools (services for which it invoiced the client). The first project mediated through SourceXchange was completed in March 2000, but just over a year later, in April 2001, the site closed down.

- Project financing through the sale of bonds. The idea behind this type of financing is similar to that of the ordinary bonds market approached by companies, but targeted at developing free software. It has several variations, but one of the best known operates as follows. When a developer (an individual or a company) has an idea for a new program, or improvement for an existing program, he writes it up as a specification, with a cost estimate for its development and issues bonds for its construction. The value of these bonds is only executed if the project is finally completed. When the developer has sold enough bonds, development begins, financed with loans based on them. When the development is completed, and an independent third party certifies that indeed what has been done complies with the specifications, the developer "executes" the bonds, settles the debts, and what is left over is the profit made from the development.

  Who would be interested in buying these bonds? Obviously, users who would want the new program or improvement to an existing program to be made. To some extent, this bonds system allows interested parties to establish developers' priorities (at least in part), through the acquisition of bonds. This also means that development costs do not have to be assumed by just one company, but rather can be shared between several (including individuals), who additionally only have to pay if the project concludes successfully in the end. A similar mechanism to this is proposed in much more detail in "The Wall Street performer protocol. Using software completion bonds to fund open source software development", by Chris Rasch (1991) [191].

Bibliography

The bonds system described is based on the *street performer protocol* ("The street performer protocol", in: *Third USENIX Workshop on Electronic Commerce Proceedings*, 1998 [152], and "The street performer protocol and digital copyrights", 1999 [153]), a mechanism based on e-commerce designed to facilitate private funding of free creations. In short, whoever is interested in a particular job would formally promise to pay a certain amount if the work is done and published as free. Its objective is to find a new way of financing relatively small jobs that are made available to everyone, but may be extended in many ways (the bonds for the construction of free software being one of them). We can see a small case of putting a derivation of this protocol into practice, the *rational street performer protocol* (Paul Harrison, 2002, [137]) where http://www.csse.monash.edu.au/~pfh/circle/funding_results.htmlit is applied to obtaining funds to finance part of The Circle, a free software project.

- Developer cooperatives. In this case, free software developers, instead of working individually or for a company, join some form of association (normally similar to a cooperative). In all other aspects, it functions the same way as a company, with an overtone of its ethical commitment to free software, which may form part of its company statutes (although an ordinary company can do this too). In this type of organisation, we may see a variety of combinations of voluntary and paid work. An example is Free Developers.

- Donations system. This involves enabling a mechanism for paying the author of a particular software, through the web page that accommodates the project. This way, users interested in the project continuing to release new versions can support it financially by making voluntary donations in the way of funding for the developer.

5.2. Business models based on free software

In addition to the project funding mechanisms that we have already talked about, another aspect related to the economy which deserves mentioning is business models. In speaking about financing mechanisms, we have already mentioned a few in passing. Here, in this section, we will describe them in a more methodical fashion.

In general, we can say that many business models are being explored around free software, some more classical and others more innovative. We need to take into account that of the most common models in the software industry it is not easy to use those based on the sale of licences, since in the world of free software this financing mechanism is very difficult to exploit. However, we can use those based on services to third parties, with the advantage that it is possible to offer complete support for a program without necessarily being its producer.

Sale of free software at so much per copy

In the world of free software it is difficult to charge for licences for use, but not impossible. In general, there is nothing in the free software definitions to prevent a company from creating a product and only distributing it to anyone who pays a certain amount. For example, a particular producer could decide to distribute its product with a free licence, but only to whoever pays 1,000 euros per copy (like in the classical world of proprietary software).

However, although theoretically this is possible, in practice it is difficult for this to occur. Because once the producer has *sold* the first copy, whoever receives it may be motivated to try and recover their investment by selling more copies at a lower price (something which cannot be prohibited by the program's licence if it is free). In the previous example, one could try selling ten copies at 100 euros each, meaning that additionally the product would work out free of charge (also, this would make it very difficult for the original producer to sell another copy at 1,000 euros, since the product could be legally obtained at a tenth of the cost). It is easy to see how this process would continue in waterfall until copies were sold at a price close to the copying marginal cost, which with current technologies is practically zero.

> Even so, and bearing in mind that the mechanism described will mean that normally a producer cannot put a price (particularly a high price) on the mere fact of the program's redistribution, there are business models that implicitly do just that. One example is the case of GNU/Linux distributions, which are sold at a much lower price in comparison with proprietary competitors, but above (and normally far above) the cost of the copy (even when it can be freely downloaded from the Internet). Of course, in these cases other factors come into play, such as the brand image or convenience for the consumer. But this is not the only case. Therefore, rather than saying that free software "cannot be sold at so much per copy", we should bear in mind that it is more difficult to do so, and that probably it will generate less profit, but that there can be models based precisely on that.

Given these limitations (and these advantages), for several years now variations on the usual business models in the software industry are being tried out, at the same time as other more innovative solutions are sought for exploiting the possibilities offered by free software. No doubt, in the next few years we will see even more experimentation in this field, and will also have more information on what models can work and under what circumstances.

In this section we offer a panorama of the business models that we most frequently encounter today, divided into groups with the intention of showing the reader what they share in common and what distinguishes them, focusing on those based on the development and services around a free software product. Revenue, in this case, comes directly from the development activities and services for the product, but does not necessarily imply new product development. When this development does occur, these models have the financing of free software products as a *subproduct*, meaning that they are particularly interesting models with a potentially large impact on the world of free software in general.

In any case, and although here we offer a relatively clear classification, we must not forget that almost all companies in reality use combinations of the models that we describe, with each other and with other more traditional ones.

5.2.1. Better knowledge

The company that follows this business model tries to make profits on its knowledge of a free product (or set of products). Its revenue will come from clients to which it will sell services related to that knowledge: development based on the product, modification, adaptation, installation and integration with other products. The company's competitive advantage will be closely related to its better knowledge of the product: therefore, the company will be particularly well positioned if it is the producer or an active participant in the project producing the software product.

This is one of the reasons why companies that use this model tend to be active participants in the projects related to the software for which they try to sell services: it is a very efficient way of obtaining knowledge about it, and more importantly, for that knowledge to be recognised. Certainly, being able to tell a

client that the company's employees include various developers on the project that produces the software, which, for example, needs to be changed, tends to provide a good guarantee.

Relationship with development projects

Therefore, companies of this type are very interested in transmitting an image of having good knowledge of certain free products. An interesting outcome of this is that support for free software projects (for example, by participating actively in them, or allowing employees to do so in the course of the working day) is not therefore, something purely philanthropic. On the contrary, it may be one of the company's most profitable assets, since clients will value it very positively as a clear sign that the company is knowledgeable about the product in question. Plus, this way it will be able to follow the development closely, trying to make sure, for example, that the improvements requested by its clients become part of the product developed by the project.

Analysing this from a more general point of view, this is a situation in which both parties, the company and the project, benefit from the collaboration. The project benefits from the development made by the company, or because some of its developers are paid (at least part-time) for their work on the project. The company benefits in knowledge about the product, image towards its clients, and a degree of influence over the project.

The range of services provided by this type of company can be very broad, but normally consists of customised developments, adaptations or integrations of the products that they are experts in, or consulting services where they advise their clients how best to use the product in question (especially if it is a complex product or its correct functioning is critical for the client).

Examples

Examples of companies that up to a point have used this business model include the following:

- LinuxCare (http://www.linuxcare.com) [45]. Established in 1996, it originally provided consulting services and support for GNU/Linux and free software in the US, and its staff consisted essentially of experts in GNU/Linux. However, in 2002 its objectives changed and since then it has specialised in providing services almost exclusively to GNU/Linux running on virtual machines in large IBM computers. Its business model has also changed to "better knowledge with limitations", since as a fundamental part of its services it offers a non-free application, Levanta.

- Alcôve (http://www.alcove.com) [3]. Established in 1997 in France, it mainly offers free software consulting services, strategic consulting, support and development. Since its foundation, Alcôve has kept the developers of various free projects on staff, trying to make a return on this in terms of image. It has also tried to offer the image, in general, of a company linked to the free software community, by collaborating, for example, with user associations and giving publicity to its collaborations with free projects (for example, through Alcôve-Labs [4]).

5.2.2. Better knowledge with limitations

These models are similar to those described in the previous section, but try to limit the competition that they may have to face. Whereas in the *pure* models based on better knowledge, anyone can, in principle, join the competition, since the software used is the same (and free), in this case the attempt is to avoid that situation by placing barriers to competition. These barriers tend to consist of patents or proprietary licences, which normally affect a small (but

fundamental) part of the developed product. This is why these models may be considered as mixed, in the sense that they are halfway between free software and proprietary software.

In many cases, the free software community develops its own version, meaning that the competitive advantage can disappear, or even turn against the company in question if the free *competitor* becomes the market standard and is demanded by the company's own clients.

Examples

There are many cases that use this business model, since it is frequently considered less risky than the *pure* knowledge one. However, the companies that have used it have evolved in different ways. Some of them include the following:

- Caldera (http://www.sco.com) [16]. Caldera's history is complicated. In the beginning, it created its own distribution of GNU/Linux, aimed at businesses: Caldera OpenLinux. In 2001 it bought the Unix division from SCO, and in 2002 it changed its name to SCO Group. Its business strategy has changed as frequently as its name, from its total support for GNU/Linux, to its legal suits against IBM and Red Hat in 2003 and abandoning its own distribution. But in relation to this heading, Caldera's business, at least until 2002, is a clear model of better knowledge with limitations. Caldera tried to exploit its knowledge of the GNU/Linux platform, but limiting the competition it could have faced by including proprietary software in its distribution. This made it difficult for its clients to change distribution once they had adopted it, because even though the other distributions of GNU/Linux included the free part of Caldera OpenLinux, they did not include the proprietary part.

- Ximian (http://ximian.com/) [74]. Founded in 1999 under the name Helix Code by developers closely connected to the GNOME project, it was acquired in August 2003 by Novell. Most of the software that it has developed has been free (in general, part of GNOME). However, in a very specific sphere Ximian decided to licence a component as proprietary software: the Connector for Exchange. This module allows one of its star products, Evolution (a personal information manager that includes e-mail, agenda, calendar, etc.,), to interact with Microsoft Exchange servers, which are commonly used by large organisations. This is how it tried to compete with an advantage over the other companies that offered services based on GNOME, perhaps with the products developed by Ximian itself but that could not interact as easily with Exchange. With the exception of this product, the Ximian model has been the one of "better knowledge", and has also been based on being the source of a program (as we will see later on). In any case, this component was released as free software in 2005.

5.2.3. Source of a free software product

This model is similar to the one based on better knowledge but with a specialisation, meaning that the company using it is the producer, almost integrally, of a free product. Naturally, the competitive advantage increases through being the developers of the product in question, controlling its evolution and having it before the competition. All of this positions the development company very strongly towards clients who are seeking services for that program. Also, it is a very interesting model in terms of image, since the company has proven its development potential by creating and maintaining the application in question, which can be very useful when it comes to convincing clients of the company's capabilities. Likewise, it creates a good image towards the free software community in general, since it receives a new free product from the company that becomes part of the common domain.

Examples

Many free products started to be developed in a company, and very often that company has continued to guide its subsequent development. Some examples:

- Ximian. We have already mentioned how it has partly used the model of better knowledge with limitations. But in general, Ximian has followed the clear model based on being the source of free programs. Its main products, like Evolution or Red Carpet, have been distributed under GPL licences. However, other also important ones, such as Mono, are distributed mostly under the MIT X11 or LGPL licences. In any case, Ximian has developed the products almost exclusively from the start. The company has tried to make a return on these developments by obtaining contracts to make them evolve in certain ways, adapting them to clients' needs, and offering customisation and maintenance.

- Zope Corporation (http://www.zope.com/) [75]. In 1995 Digital Creations was established, developing a proprietary product for the management of classified ads on the web. In 1997 it received a capital injection from, among others, a venture capital company called Opticality Ventures. What was strange about this investment (at that time) was the condition that was imposed of distributing the evolved product as free software, which later became Zope, one of the most popular content managers on the Internet. Since then, the company's business model has been to produce Zope and related products, and to offer adaptation and maintenance services for all of them. Zope Corporation has also known how to create a dynamic community of free software developers around its products and to collaborate actively with them.

5.2.4. Product source with limitations

This model is similar to the previous one, but takes measures to limit the competition or to maximise revenue. Among the most common limitations, we can find the following:

- Proprietary distribution for a time, then released as free software. With or without a promise of a later free distribution, each new version of the product is sold as proprietary software. After a certain amount of time (normally, when a new version is released, also as proprietary software), the old version is distributed with a free licence. This way, the production company obtains revenue from clients interested in having the new versions, and at the same time limits the competition, since any company wanting to compete using that product can only do so with the free version (only available when the new proprietary version is released, which is supposedly improved and more complete).

- Limited distribution for a period. In this case, the software is free as of the moment it is first distributed. But because there is nothing in the free licence forcing to distribute the program to anyone who wants it (this is something that the person in possession of the software may or may not do), the producer distributes for a time to its clients only, who pay for it (normally in the form of a maintenance contract). After a while, it distributes it to anyone, for example by placing it in a public access file. This way, the producer obtains income from its clients, who perceive this preferential availability of the software as an added value. Naturally, the model only works if the clients do not in turn make the program public when they receive it. For certain types of clients, this may not be common.

In general, in these cases the development companies obtain the mentioned benefits, but not at zero cost. Because of the delay with which the product is available for the free software community, it is practically impossible for it to be able to contribute to its development, meaning that the producer will benefit very little from external contributions.

Examples

Some companies that use this business model are as follows:

- artofcode LLC (http://artofcode.com/) [9]. Since the year 2000, artofcode sells Ghostscript in three versions (previously Alladin Enterprises had done this with a similar model). The latest version is distributed as AFPL Ghostscript, under a proprietary licence (which allows use and non-commercial distribution). The next one (with a year's delay more or less) is distributed as GNU Ghostscript, under the GNU GPL. For example, in summer 2003, the AFPL version is 8.11 (released on 16th August), while the GNU version is 7.07 (distributed as such on 17th May, but whose equivalent AFPL version is dated 2002). Also, artofcode offers a third version, with a proprietary licence that allows its integration with products not compatible with the GNU GPL (in this case it uses a dual model, which we will describe later on).

- Ada Core Technologies (http://www.gnat.com/) [2]. It was established in 1994 by the authors of the first Ada 95 compiler, developed with partial funding from the US Government, based on GCC, the GNU compiler. Since the beginning its products have been free software. But most of them are first offered to their clients, as part of a maintenance contract. For example, its compiler, which continues to be based on GCC and is distributed under the GNU GPL, is offered to its clients as GNAT Pro. Ada Core Technologies does not offer this compiler to the general public by any means, and normally you cannot find versions of it on the Net. However, with a varying delay (of about one year), Ada Core Technologies offers the *public* versions of its compiler, very similar but without any type of support, in an anonymous FTP file.

5.2.5. Special licences

Under these models, the company produces a product that it distributes under two or more licences. At least one of them is free software, but the others are typically proprietary and allow the product to be sold in a more or less traditional way. Normally, these sales are complemented with the sale of consulting services and developments related to the product. For example, a company can distribute a product as free software under the GNU GPL, but also offer a proprietary version (simultaneously, and with no delay between the two versions) for those not wanting the conditions of the GPL, for example, because they want to integrate the product with a proprietary one (which the GPL does not allow).

Example

Sleepycat Software (http://www.sleepycat.com/download/oslicense.html) [60]. This company was established in 1996 and has announced that it has made a profit from the start (which is certainly remarkable in a software- related company). Its products, including Berkeley DB (a very popular data manager because it can be easily embedded in other applications), are distributed under a free licence that specifies that in the case of embedding with another product, it has to provide the source code of both. Sleepycat offers consulting and development services for its products, but also offers them under licences that allow them to be embedded without having to distribute the source code. Of course, it does this under a specific contract, and in general, under a proprietary software sales regime. In 2005, Sleepycat Software was bought by Oracle.

5.2.6. Brand sale

Although it is possible to obtain very similar products for far less money, many clients are prepared to pay extra to buy a *brand*. This principle is adopted by companies that invest in establishing a brand with a good and well-recognised image that allows them to then sell free products with a sufficient margin. In many cases, they do not just sell those products, but also services that the clients will also accept as an added value.

The most well-known cases of this business model are the companies that sell GNU/Linux distributions. These companies try to sell something that in general can be obtained at a far lower cost from the Net (or others sources with less of a brand image). Therefore, they have to make consumers recognise their brand and be prepared to pay the additional cost. To do so, they don't just invest in publicity, they also offer objective advantages (for example, a well-assembled distribution or a distribution channel that offers proximity to the client). Also, they tend to offer a large number of services around it (from training to third party certification programs), in order to make the most of the brand image.

Example

Red Hat (http://www.redhat.com) [56]. Red Hat Linux started to be distributed in 1994 (the company started to be known by its current name in 1995). For a long time, Red Hat managed to establish its name as the GNU/Linux distribution par excellence (although in the mid 2000 it shares that position with other companies like OpenSUSE, Ubuntu, and perhaps Debian). Several years down the line Red Hat sells all types of services related to the distribution, GNU/Linux and free software in general.

5.3. Other business model classifications

Free software literature provides other classifications of traditional business models. As an example, here are a few.

5.3.1. Hecker classification

The classification provided in "Setting up shop: the business of open source software" (Frank Hecker, 1998) [141] was most used in the publicity of the Open Source Initiative, and also one of the first to try and classify the businesses that were emerging around that time. However, it includes various models that have little to do with free software (where free software is little more than a companion to the main model). In any case, the models it describes are as follows:

- *Support seller* (sale of services related to the product). The company promotes a free software product (which it has developed or in which it participates actively) and sells services such as consulting or adaptation to specific requirements.

- *Loss leader* (sale of other proprietary products). In this case, the free program is used to somehow promote the sale of other proprietary products related to it.

- *Widget frosting* (Sale of hardware). The main business is the sale of hardware and the free software is considered a complement that can help the company obtain a competitive advantage.

- *Accessorising* (sale of accessories). Products related to free software are sold, such as books, computer devices, etc.

- *Service enabler* (sale of services). The free software serves to create a service (normally accessible online) from which the company makes a profit.

- *Brand licensing* (sale of a brand). A company registers trademarks that it manages to associate with free software programs, probably that it has developed itself. Then it obtains income through licensing the use of those trademarks.

- *Sell it, free it*. This is a similar model to the *loss leader*, but done in a cyclical fashion. First a product is marketed as free software. If it is relatively successful, the next version is distributed as proprietary software for a time, after which it is freed. By then, a new proprietary version is being distributed, and so on successively.

- *Software franchising*. A company franchises the use of its brands in relation to a particular free program.

5.4. Impact on monopoly situations

The software market tends towards the domination of one product in each of its segments. Users want to make the most of the effort made in learning how a program works, companies want to recruit people who are familiar with the use of their software, and everyone wants the data that they handle to be manageable by the programs of the companies and people with whom they work. This is why any initiative designed to break a *de facto* situation in which one product clearly dominates the market is destined to produce more of the same: if it is successful, the new product will come to take its place, and

Note

Readers will have observed that this classification is fairly different to the one that we have given, but even so some of the categories almost totally match some of ours.

in a short period we will have a new dominant product. Only technological changes produce, during a short period, sufficient instability for nobody to dominate clearly.

But the fact that there is a dominant product does not necessarily have to lead to the creation of a business monopoly. For example, petrol is a product that almost dominates the fuel market for private cars, but (in a free petrol market) there are many production companies and distribution companies for that same product. In reality, when we talk about software, what is worrying is what happens when a product manages to dominate the market because that product has a sole possible supplier. Free software offers an alternative to that situation: free products can be promoted by a specific company, but that company does not control them, or at least not to the extent that proprietary software has us accustomed to. In the world of free software, a dominant product does not necessarily entail the monopoly of one company. On the contrary, irrespective of the product that dominates the market, many companies can compete in providing it, improving it, adapting it to clients' needs and offering services related to it.

5.4.1. Elements that favour dominant products

In computer software, it is common to have a clearly dominant product in each market segment. And this is normal for several reasons, among which we would highlight the following:

- Data formats. In many cases the data format is very closely linked to an application. When a sufficiently high number of people uses it, the data format becomes the *de facto* standard, and the pressures to adopt it (and therefore, the application) are tremendous.

- Distribution chains. Normally, one of the problems with starting to use a program is obtaining a copy of it. And it is normally difficult to find programs that are not leaders in their market. Distribution chains are expensive to maintain, meaning that it is difficult for minority competitors to reach the computer shop where the end user can buy them. However, for the dominant product it is easy: the first to be interested in having it will be the computer shop itself.

- Marketing. The "free" marketing that a product obtains once a significant proportion of the population uses it is enormous. "Word of mouth" also works very well when we ask and exchange information with the people we know. But above all the impact from the media is enormous: computer magazines will refer time and again to a product if it appears to be the one used the most; there will be training courses around it, books describing it, interviews with users, etc.

- Investment in training. Once time and money has been spent on learning how a tool functions, there is a high motivation not to change that tool. Also, that tool is usually the one that already dominates the market, because it is easier to find people and materials to help teach how to use it.

- Pre-installed software. Receiving a machine with pre-installed software is certainly a great incentive towards using it, even if it has to be paid for separately. And normally, the type of software that the seller of the machine will be prepared to pre-install will only be the most used.

5.4.2. The world of proprietary software

In the world of proprietary software the appearance of a dominant product in any segment is equivalent to a monopoly on the part of the company that produces it. For example, we have these *de facto* monopoly situations (or almost) of a product and a company in the market for operating systems, desktop publishing, databases, graphic design, text processors, spreadsheets, etc.

And this is so because the company in question has enormous control over the leading product, so much so that only they can drive its evolution, the fundamental lines along which it will be developed, its quality, etc. Users have very little control, since they have very little motivation to consider other products (for the reasons we have mentioned in the preceding section). In view of this, there is little that competition can do, except to try and defy the product's dominant position by improving their own products, (to try and counteract those very reasons), normally with limited success.

This situation places the entire sector in the hands of the dominant company's strategy. All of the actors depend on it, and even the development of software technology in that field will be mediatised for the improvements that it makes to its product. In general terms, this is a situation where the worst economic effects of a monopoly arise, and in particular, the lack of motivation for the dominant company to tailor products to the (always evolving) needs of its clients, as they have become a captive market.

5.4.3. The situation with free software

However, in the case of free software a dominant product does not automatically translate into a business monopoly. If the product is free, any company can work with it, improve on it, adapt it to clients' needs, and in general, help it to evolve. Also, precisely due to its dominant position, there will be many companies interested in working with it. If the "original" producer (the company that originally developed the product) wishes to remain in the business, it will have to compete with all of them and will therefore be highly motivated

to make its product evolve precisely along the lines that users want. Of course, it will have the advantage of better knowledge of the program, but that isn't all. They will have to compete for every client.

Therefore, the appearance of dominant products in the world of free software, translates into more competition between companies. And with it users recover control: companies in competition cannot do anything but listen to them if they want to survive. And this is precisely what will make sure that the product improves.

> **Free products that are dominant in their sector**
>
> For a long time, Apache has been the leader in the market for web servers. But there are many companies behind Apache, from some very large ones (like IBM) to other much smaller ones. And all of them have no other choice but to compete by improving it and normally by contributing to the project with their improvements. Despite the fact that Apache is almost a monopoly in many fields (for example, it is almost the only web server considered on the GNU/Linux or *BSD platform), it does not depend on a single company, but rather on literally dozens of them.
>
> The distributions of GNU/Linux are also an interesting case. GNU/Linux is not, certainly, a monopoly, but is possibly the second choice in the market for operating systems. And this has not forced a situation whereby one company has control over it. On the contrary there are tens of distributions made by different companies, which freely compete in the market. Each one of them tries to offer improvements, which its competitors have to adopt at the risk of being left out. Moreover, they cannot stray too far from what is the "GNU/Linux standard", since this would be rejected by users as a "departure from the norm ". The situation after several years of a growing market share for GNU/Linux shows us tens of companies that compete and allow the system to evolve. And once again, all of them pursue satisfying users' requirements. This is the only way that they can stay in the market.
>
> GCC is a dominant product in the world of C and C++ compilers for the GNU/Linux market. And yet, this has not led to any company monopoly situation, even though Cygnus (now Red Hat) was responsible for a long time for coordinating its development. There are many companies that make improvements to the system and all of them compete, each in their specific niche, to satisfy their users' demands. In fact, when a specific company or organisation has failed in the task of coordinating (or some users have perceived this to be the case) there has been room for the project to *fork*, with two products running in parallel for a while, until they have come back together again (as is now happening with GCC 3.x).

5.4.4.   Strategies for becoming a monopoly with free software

Despite the fact that the world of free software is much more hostile to business monopolies than the world of proprietary software, there are strategies that a company can use to try to approach a situation of monopolistic dominance of a market. These practices are common in many other economic sectors and in order to prevent them we have bodies that regulate competition, which is why we will not go into too much detail about them. However, we will mention one that, up to a point, is specific to the software market, and which has already been experienced in certain situations: the acceptance of third party product certification.

When a company wishes to distribute a software product (free or proprietary) that functions in combination with others, it is common to "certify" that product for a certain combination. The manufacturer undertakes to offer services (updates, support, problem-solving, etc.) only if the client guarantees that the

product is being used in a certified environment. For example, the manufacturer of a database management program can certify its product for a certain GNU/Linux distribution, and no other. This implies that its clients will have to use that GNU/Linux distribution or forget having the manufacturer's support (which, if the product is proprietary may be impossible in practice). If a particular manufacturer manages to achieve a clearly dominant position as a third-party certified product, users are not going to have any other choice than to use that product. If in that segment certification is important, we will once again be facing a business monopoly situation.

Note

Up to a point, in the market for GNU/Linux distributions we are starting to see a few cases of situations tending towards a *de facto* monopoly through certification. For example, there are many manufacturers of proprietary products that only certify those products on a given GNU/Linux distribution (very commonly Red Hat Linux). For the time being this is not resulting in a monopoly situation for any company, which may be due to the fact that certification is not so relevant for users in the market for GNU/Linux distributions. But only the future will tell if at some point this situation approaches a *de facto* monopoly.

Nonetheless, it is important to bear in mind two comments in relation to the above. The first is that these monopoly positions will not be easy to achieve, and in any case will be achieved through "non-software" mechanisms in general (unlike the dominant product situation, which as we have seen is relatively normal, reached through mechanisms purely related to IT and its patterns of use). The second is that if all the software used is free, that strategy has limited chances of succeeding (if any at all). A manufacturer may manage to get lots of companies to certify for its products, but clients will always be able to look to different companies for services and support other than those that have certified for it, if they consider it appropriate.

6. Free software and public administrations

"[...] for software to be acceptable for the State, it does not only need to be technically capable of performing a task, but also its contracting conditions need to meet a series of requirements regarding licensing, without which the State cannot guarantee to its citizens that their data is being adequately processed, with due regard for confidentiality and accessibility over time, because these are highly critical aspects of its normal duty."

Edgar David Villanueva Núñez (letter of reply to the general manager of Microsoft Peru, 2001)

Public institutions, both those with the capacity to legislate and those dedicated to administrating the State (the "public administrations"), play a very important role where adopting and promoting technologies is concerned. Although up to 2,000 of these institutions showed practically no interest in the free software phenomenon (save for anecdotal cases), the situation started changing as of then. On the one hand, many public administrations started using free software as part of their IT infrastructure. On the other hand, in their role as promoters of the information society, some started to promote directly or indirectly the development and use of free software. Also, some legislative bodies have started paying attention (bit by bit) to free software, sometimes favouring its development, sometimes impeding it, and sometimes just taking its presence into consideration.

Before going into detail, it is important to remember that for a long time free software was developed without explicit backing (or even interest) from public institutions. For this reason, the recent attention that it is drawing from many of them is not without controversy, confusion and problems. Also, in the last few years initiatives related to open standards are gaining momentum, often resulting in measures (more or less directly) associated to free software.

In this chapter we will try to describe the current situation and the peculiarities of free software in relation to the "public" sphere.

6.1. Impact on the public administrations

Several studies have been made focusing on the use of free software in public administrations (for example, "Open source software for the public administration", 2004 [159]; "Open source software in e-Government, analysis and recommendations drawn up by a working group under the Danish board of technology", 2002 [180]; "Free software / open source: information society opportunities for Europe?", 1999 [132], and "The case for government promotion of open source software", 1999 [213]). Next, we will discuss some of the most notable ones (both positive and negative).

6.1.1. Advantages and positive implications

Some of the advantages of using free software in public administrations and the main new prospects that it offers are as follows: **1)** Developing local industry

One of the major advantages of free software is the possibility of developing a local software industry. When we use proprietary software, everything spent on the licences goes directly to the product's manufacturer, and the purchase strengthens the manufacturer's position, which is not necessarily negative, but is not very efficient for the region to which the Administration is associated when we consider the alternative of using a free program.

In this case, local companies will be able to compete in providing services (and the program itself) to the Administration, under very similar conditions to any other company. Let's say that somehow the Administration is levelling the playing field and making it easier for anyone to compete on it. And of course, that "anyone" includes local companies, who will have the opportunity to exploit their competitive advantages (better knowledge of the client's needs, geographical proximity, etc.).

2) Independence from a supplier and market competition

Obviously, any organisation will prefer to depend on a competitive market than on a single provider capable of imposing the conditions under which it supplies its product. However, in the world of the Public Administration, this preference becomes a basic requirement, and even a legal obligation in some cases. In general, the Administration cannot choose to contract a given supplier, but rather must specify its requirements in such a way that any interested company that fulfils certain characteristics and that offers the required product or service, can opt for a contract.

Once again, in the case of proprietary software, each product has just one supplier (even if it uses a number of intermediaries). If a particular product is specified, then the Administration will also be deciding what provider to award the contract. And in many cases it is virtually impossible to avoid specifying a particular product, when we are dealing with computer programs. Reasons of compatibility within the organisation or savings in training and administration, or many more, make it common for an administration to decide to use a certain product.

The only way out of this situation is by making the specified product free. This way, any interested company will be able to supply it and also any type of service related to it (subject only to the company's capabilities and knowledge of

the product). Also, in the case of this type of contracting, the Administration can change supplier in the future if it wishes, and without any technical problems, since even if it changes company, it will still be using the same product.

3) Flexibility and adaptation to specific requirements

Although adaptation to specific requirements is something that any organisation using computers needs, the peculiarities of the Administration make this a very important factor in the successful implantation of a software system. In the case of free software, the adaptation is made much easier, and more importantly, can rely on a competitive market if contracting it is necessary.

When the Administration buys a proprietary product, modifying it normally involves reaching an agreement with the manufacturer, who is the only party that can legally (and often technically) do it. Under these circumstances, it is difficult to negotiate, especially if the manufacturer is not excessively interested in the market offered by that particular administration. However, by using a free product, the Administration can modify it as it wishes, if it employs capable personnel, or outsource the modification. In principle, this outsourcing is possible with any company that has the skills and knowledge to do so, meaning that several companies can be expected to compete. Naturally, this tends to make the cost cheaper and improve the quality.

> **The case of GNU/Linux distributions**
>
> In the last few years in Spain, it has become common for certain autonomous communities to create their own GNU/Linux distributions. This trend started with GNU/Linux, but nowadays there are many more. Although some experts have criticised the existence of these distributions, it is a clear example of the flexibility that free software allows. Any public administration, by spending relatively moderate resources, can contract a GNU/Linux adaptation adapted to its needs and preferences, without practically any limits. For example, it can change the desktop appearance, choose the set of default applications and language, improve the applications' localisation, etc. In other words: if wanted, the desktop (and any other software element that works on the computer) can be adapted to precise requirements.
>
> Of course, this adaptation will involve some expenditure, but experience shows that it can be achieved relatively cheaply, and the trend appears to indicate that it will be increasingly easier (and cheaper) to make customised distributions.

4) Easier adoption of open standards

Given their very nature, free programs commonly use open or non-proprietary standards. In fact, almost by definition, any aspect of a free program that we may care to consider can be reproduced easily and, therefore, is not proprietary. For example, the protocols used by a free program in order to interact with other programs can be studied and reproduced, meaning that they are not proprietary. But also, quite commonly and in the interest of the projects themselves, we try to use open standards.

In any case, irrespective of the motive, it is a fact that free programs normally use non-proprietary standards for data exchange. The advantages of this for public administrations are more far-reaching than for any other organisation, since the promotion of proprietary standards (even indirectly, by using them) is much more of a concern in their case. And in at least one aspect, the use of non-proprietary standards is fundamental, where interaction with citizens is concerned, since they should not be forced to purchase any product from a particular company in order to be able to interact with the Administration.

5) Public scrutiny of security

For a public administration, being able to guarantee that its computer systems only do what they have to is a fundamental obligation, and in many countries, a legal requirement. Often these systems handle private data, which third parties could be interested in (for example tax data, criminal records, census or electoral data, etc.). If a proprietary application is used, without source code available, it is difficult to guarantee that the application will process the data in the way that it should. But even if it does provide its source code, the possibilities of a public institution ensuring that it does not contain *strange* code will be very limited. Only if the task can be habitually and routinely commissioned to third parties, and plus any interested party can scrutinise it, can the Administration be reasonably sure that it is complying with its fundamental duty, or at least taking the measures within its power to do so.

6) Availability in the long term

Much of the data processed by the administrations, and the programs used to calculate them, need to be available within decades and decades. It is very difficult to guarantee that any proprietary program will be available after this time, especially if the idea is for it to work on the usual platform at that time in the future. On the contrary, it is possible that the manufacturer may have lost interest in the product and has not ported it to new platforms, or is only prepared to do so for a lot of money. Once again, we need to remember that only the manufacturer can port the product, meaning that negotiations will be difficult. In the case of free software, however, the application is available, with certainty, so that anyone can port it and leave it functioning according to the needs of the Administration. If this does not happen spontaneously, the Administration can always look for several companies to make the best offer to do the job. This guarantees that the application and the data that it processes will be available when needed.

7) Impact beyond use on the part of the Administration

Many applications used or promoted by the public administrations are also used by many other sectors of society. For this reason, any public investment in the development of a free product benefits not only the Administration itself, but also all its citizens, who will be able to use the product for their computer tasks, perhaps with the improvements made by the Administration.

Bibliography

Readers interested in a report on the advantages of free software for the Administration, written in the US context of 1999, can consult "The case for government promotion of open source software" (Mitch Stoltz, 1999) [213].

Note

A very particular case, but one with enormous impact, which displays this better use of public resources is program localisation (adaptation to a community's uses and customs). Although the most visible aspect of localisation is the translation of the program and its documentation, there are others that are also affected by it (from use of the local currency symbol to presenting the date and time in the formats of the community in question, to the use of examples in the documentation and ways of expression adapted to local customs).

In any case, obviously if a public administration uses funds to localise a particular application tailoring the application to its needs, it is more than likely that those needs coincide with those of its citizens, meaning that it will generate, not only a program that satisfies its own requirements, but also, one that can be made available to any citizen able to make the most of it at no additional cost. For example, when an administration finances the adaptation of a computer program to a language that is used within its community, it will not only be able to use that program within its own offices, but also offer it to citizens, with everything that this involves in terms of developing the information society.

6.1.2. Difficulties of adoption and other problems

However, although there are many advantages for the administration using free software, there are also many difficulties that need to be faced when it comes to putting it into practice. Of them, we would particularly mention the following:

1) Lack of knowledge and political commitment
   The first problem that free software encounters for entering the Administration is one that other organisations undoubtedly share: free software is still unknown for the people who make the decisions.
   Fortunately, this is a problem that is gradually being solved, but in many spheres of the Administration, free software is still perceived as something *strange*, so decisions about using it still involve certain risks.
   In addition to this, we tend to come across a problem of political decision-making. The main advantage of free software for the Administration is not the cost (since the cost, in any case, is high, especially when we are talking about a roll-out for a large number of workstations), but as we have already said, benefits are above all strategic. And therefore, the decision falls within the political, rather than the technical sphere. Without the political will to change software systems and the philosophy with which they are contracted, it is difficult to progress with the deployment of free software in the Administration.

2) Poor adaptation of contracting mechanisms
   The contracting mechanisms that the Administration uses nowadays, ranging from the usual public tender models to cost itemising, are fundamentally designed for the purchase of IT products and not so much for

the purchase of services related to programs. However, when we use free software, normally there is no product to be bought, or its price is negligible. In contrast, to take advantage of the opportunities provided by free software, it is convenient to be able to contract services around it. This makes it necessary, before free software can be seriously used, to design bureaucratic mechanisms that facilitate contracting in these cases.

3) Lack of deployment strategy

Often an administration may start to use free software simply because the purchase cost is lower. It is common in these cases for the product in question to be incorporated into the computer system with no further planning, and in general, without a global strategy for using and making the most of the free software. This causes most of its benefits to be lost along the way, since everything boils down to the use of a *cheaper product*, whereas we have already seen that, in general, the major benefits are of a different type.

If added to this, the transition is not properly designed, the use of free software can incur considerable costs, and we will see that in certain isolated cases, outside of a clear framework, the use of free software in the Administration can be unsuccessful and frustrating.

4) Scarcity or lack of free software products in certain segments

The deployment of free software in any organisation can encounter the lack of free quality alternatives for certain types of applications. For these cases, the solution is complicated: all that we can do is try to promote the appearance of the free product that we need.

Fortunately, public administrations are in a good position to study seriously whether they may be interested in promoting or even financing or co-financing, the development of that product. We should remember that its objectives normally include providing its citizens with better access to the information society, for example, or promoting the local industrial fabric. Certainly, the creation of many free programs will have a positive influence on both objectives, meaning that we should add to the mere direct cost/benefit calculation, the indirect benefits that such a decision will have.

5) Interoperability with existing systems

It is not common for a full migration to free software to be made with the entire system at the same time. Therefore, it is important for the part that we want to migrate to continue functioning correctly in the context of the rest of the software with which it will have to interoperate. This is a well-known problem with any migration (even if it is a proprietary product), but it can have a particular impact in the case of free software. In any case, it will be something to be taken into account when studying the transition. Fortunately, we can often adapt the free software that needs to be in-

stalled to interoperate adequately with other systems, but if this is needed, this point will have to be considered when budgeting the migration costs.

6) Data migration

This is a generic problem of any migration to new applications that use different data formats, even if they are proprietary. In fact, in the case of free software this problem is often mitigated, since it is usual to make a special effort to foresee as many formats and data exchange standards as possible. But normally the data has to be migrated. And the cost of doing this is high. Therefore, in calculating the cost of a potential migration to free software, this factor needs to be carefully considered.

6.2. Actions of the public administrations in the world of free software

Public administrations influence the world of software in at least three ways:

*   By buying programs and services related to them. Administrations, as large users of software, are fundamental players in the software market.

*   By promoting different ways of using (and purchasing) certain programs by individuals or companies. This promotion is sometimes achieved by offering financial incentives (tax deductions, direct incentives, etc.), sometimes through information and advice, sometimes by "follow my example"...

*   By financing (directly or indirectly) research and development projects that design the future of software.

In each of these spheres free software can offer specific advantages (in addition to those already described in the preceding section) of interest to both the Administration and to society in general.

6.2.1. How to satisfy the needs of the public administrations?

Public administrations are large consumers of IT. Where software is concerned, they normally buy off-the-shelf products as well as customised systems. From this point of view, they are fundamentally large purchasing centres, similar to those of big companies, but with their own peculiar features. For example, in many spheres, the purchasing decisions of the public administrations are supposed to take into consideration not only cost versus functionality parameters, but also others, such as the impact of the purchase on the industrial or social fabric or long term strategic considerations, which can also be important.

In any case, the usual nowadays with off-the-shelf software is to use market leader proprietary products. The amount of public money spent by municipalities, regional and national governments, and international (such as European Union) public administrations on purchasing Windows, Office or other similar product licences is certainly considerable. But gradually free solutions are starting to enter the market. Increasingly, solutions based on free software are being considered for servers, and products such as OpenOffice.og, and GNU/Linux with GNOME or KDE are increasingly used for the desktop.

What is there to be gained from this migration to free software? To illustrate just what, let's consider the following scenario. Let's suppose that with a fraction of what is spent on two or three "star" proprietary products by all the European administrations (or probably those of any medium-sized developed country), we could convene a public tender for one company (or two, or three, or four) to improve and adapt the currently available free programs so that within one or two years they would be ready for massive use, at least for certain standard tasks (if they are not already). Let's imagine for example, a coordinated effort, on a national or European scale, whereby all the administrations participated in a consortium responsible for managing these tenders. In a short period of time there would be a "local" industry specialised in making these improvements and adaptations. And the administrations could choose between the three or four free distributions produced by that industry. In order to promote competition, each company could be compensated according to the number of administrations that chose to use their distribution. And the entire result of this operation, because it would be free software, would also be available for companies and individual users, which in many cases would have similar needs to the administrations'.

In the case of customised software, the normal process currently involves contracting the necessary programs from a company under a proprietary model. Any development made at the Administration's request is the property of the company that develops it. And usually, the contracting administration is tied to the supplier in everything related to improvements, updates, and support, in a vicious circle that makes competition difficult and slows down the process of modernising public administrations. Even worse, is that often the same program is sold time and again to similar administrations, applying in each case the costs incurred for making the development from scratch.

Let's consider again how things could be different. A consortium of public administrations needing a particular type of customised software could demand that the obtained result be free software. This would allow other administrations to benefit from the work and in the medium term may interest them in collaborating in the consortium so that their particular requirements could be taken into consideration. Because the resulting software would be free, there would be no obligation to contract the improvements and adaptations to the same supplier, meaning that competition would enter the market (which at

present is almost captive). And in all events, the final cost for any of the administrations involved would never be more than if a proprietary model had been adopted.

Are these scenarios science fiction? As we will see later, there are timid initiatives in similar directions to the ones described. In addition to helping to create and maintain an industry within the sphere of the purchasing public administration, free software offers more specific advantages in the public domain. For example, it is the most efficient way of having software developed in minority languages (a basic concern of many public administrations). It can also help a lot towards maintaining strategic independence in the long term and ensuring the accessibility of the data in public administrations' custody for a long time. For all of these reasons, public bodies are increasingly interested in free software as users.

Some cases related to German administrations

In July 2003 the first stable version of Kolab was released, a product of the Kroupware project. Kolab is a free IT help system for group work (*groupware*) based on KDE. The reason for mentioning this project is that originally it was a tender by the German government's Bundesamt für Sicherheit in der Informationstechnik (BSI - translated as the Federal Office for Information Security). This tender sought a solution that would interoperate with Windows and Outlook on the one hand, and GNU/Linux and KDE on the other. Of the submitted bids, the joint proposal of three companies, Erfrakon, Intevation and Klarälvdalens Datakonsult, was awarded the contract, with their proposal to provide a free solution partly based on software already developed by the KDE project, completed with its own free developments, resulting in Kolab.

In May 2003, the Town Hall of Munich (Germany) approved the migration to GNU/Linux and free office suite applications for all desktop computers, about fourteen thousand in total. The decision to do this was not purely financial: strategic and qualitative aspects were also taken into consideration, according to the authorities. In the comprehensive analysis that was carried out prior to making the decision, the solution that was finally chosen (GNU/Linux plus OpenOffice, fundamentally) obtained 6,218 points (from a maximum of ten thousand) as opposed to the little more than five thousand points obtained by the "traditional" solution based on Microsoft software.

In July 2003, the Koordinierungs-und Beratungsstelle der Bundesregierung für Informationstechnik in der Bundesverwaltung (KBSt), under the German Ministry of the Interior, made public the document *Leitfaden für die Migration von Basissoftwarekomponenten auf Server- und Arbeitsplatzsystemen* [107] ('Migration guide for the basic software components of servers and workstations'), which offers German public bodies a set of guidelines on how to migrate to solutions based on free software. These guidelines are designed for the decision-making party to evaluate whether a migration to free software is appropriate and how to carry out the migration if that decision is made.

6.2.2. Promotion of the information society

Public bodies spend a lot of resources on incentives to encourage IT spending. This is a formidable tool, which can help new technologies to expand in society. But it is also a dangerous tool. For example, it may not be a very good idea to promote society's use of the Internet by recommending a particular navigator encouraging one company's *de facto* monopoly position, because in the long term this could be negative for the society that we are trying to benefit.

Once again, free software can help in these situations. In the first place, it is neutral towards manufacturers, since nobody has the exclusivity over any free program. If an administration wishes to promote the use of a family of free programs, it can convene a tender, which any company in the sector can bid for, to manage its delivery to citizens, its improvement or extended functionality, etc. Secondly, it can help a lot in economic aspects. For example, in many cases the same amount of funds can be spent on purchasing a certain number of licences for proprietary programs as for purchasing one free copy and contracting support or adaptations for it; or even on negotiating with a proprietary software manufacturer for the rights to convert its product into free software.

In a separate field, we could imagine dedicating part of the amount allocated for the computerisation of schools to creating a GNU/Linux distribution adapted to primary schools' teaching requirements. And with the rest of the funds contracting support for maintaining the software in those schools, so that the software is not merely "for show" but rather people are genuinely responsible for ensuring that it works correctly. This not only covers educational requirements but also generates a market for companies, usually local ones, capable of providing maintenance services. And of course, it leaves the path to the future completely open: the software will not become obsolete in just a few years meaning that we need to start over from scratch, rather it can be updated incrementally, year after year, maintaining the program's benefits with a similar investment.

Note

Readers who are familiar with public initiatives in respect of free software will recognise the case of gnuLinEx in this example. Towards the end of 2001, the Regional Government of Extremadura (Spain) decided to use a GNU/Linux distribution in order to computerise all of the public schools in the region. To do so, it financed the construction of gnuLinEx, a GNU/Linux distribution based on Debian that was announced in spring 2002, and made sure that it was a requirement in all tenders for purchasing schools' computer equipment. Also, it started training programs for teachers, creating teaching materials and expanding the experience into other fields. In mid- 2003, it seemed that the experience was a success, as it expanded institutionally to other regions (for example, to Andalucía, also in Spain, through the Guadalinex project).

6.2.3. Research promotion

Free software also provides noteworthy benefits where R+D policies are concerned. Public money is being used to finance a large amount of software development that society does not end up benefiting from, even indirectly. Usually, public research and development programs finance, wholly or in part, projects to create software without really worrying about the rights that the public will have over them. In many cases the results, without an adequate commercialisation plan, are simply filed and left to gather dust. In others, the same people who financed a program through taxes end up paying for it again if they wish to use it (since they need to buy licences for use).

Free software offers an interesting choice, which the authorities responsible for innovation policies in many administrations are gradually starting to consider with care. Especially when the research is pre-competitive (most common in the case of public funding), the fact that resulting programs are free allows industry as a whole (and consequently society) to benefit enormously from the public money spent on R+D in the software field. Where one company may see a result that is impossible to sell, another may see a business opportunity. This way, on the one hand, the results of research programs are maximised, and on the other, competition between companies wishing to use the results of a project increases, since all of them will compete on the basis of the same programs resulting from the project.

This model is not new. To a great extent it is the one that has allowed the Internet to develop. If public administrations demand that the results of research carried out with its funds is distributed in the form of free software, it would not be surprising for similar cases to appear, on different levels. Either the outcome of that research will be poor or useless (in which case the way of selecting funding projects needs to be reviewed), or the dynamic generated by leaving them ready for any company to be able to convert them into a product would allow simply unforeseeable developments.

6.3. Examples of legislative initiatives

In the following sections we look at some specific legislative initiatives relating to the use and promotion of free software by public administrations. Of course, the list we provide does not intend to be exhaustive, and has focused on the initiatives that have been pioneering in some way (even if they were not finally approved). Interested readers can complete it by consulting "GrULIC. Legislation regarding the use of free software by the State" [133], which cites many more similar cases. Also, in one appendix (appendix D) we include for illustrative purposes the full text or the most relevant parts of several of these initiatives.

6.3.1. Draft laws in France

In 1999 and 2000 in France two draft laws related to free software were presented, which were pioneers in a long series of legislative debates over the issue:

- Draft law of 1999-495, proposed by Laffitte, Trégouet and Cabanel, was made available on Senate of the French Republic's web server in October 1999. Following a process of public debate over the Internet (http://www.senat.fr/consult/loglibre/index.htm) [102] which lasted two months, the draft was modified. The result was Draft Law number 2000-117 (Laffitte, Trégouet and Cabanel, Proposition de Loi numéro 117, Senate of the French Republic, 2000) [162], which advocated the obligatory use of free software by the Administration, contemplating exceptions

and transition measures for cases where it was not yet technically possible, in the more general context of expanding the use of the Internet and free software across the French administration.

- In April 2000, members of parliament Jean-Yves Le Déaut, Christian Paul and Pierre Cohen proposed a new law whose objective was similar to that of Laffitte, Trégouet and Cabanel's draft: to reinforce the freedoms and security of consumers, in addition to improving the equality of rights in the information society.

  However, unlike the draft law of Laffitte, Trégouet and Cabanel, this second one did not make it compulsory for the Administration to use free software. This draft law centred on the fact that the software used by the Administration should have the source code available, but without forcing it to be distributed with free software licences.

  In order to achieve their objectives, the legislators aimed to guarantee the software's "right to compatibility", by providing mechanisms that put into practice the principle of interoperability reflected in EC Directive related to the legal protection of computer programs (Council Directive 91/250/EEC, of 14th May 1991, regarding the legal protection of computer programs, 1991) [111].

Neither of the two French drafts was passed into law, but both have served to inspire most subsequent initiatives worldwide, which is why they are particularly interesting to study. The second one (proposed by Le Déaut, Paul and Cohen) pursued the compatibility and interoperability of the software, emphasising the availability of the source code for the software used by the Administration. However, it did not require developed applications to be free software, understood as meaning software distributed under licences that guarantee the freedom to modify, use and redistribute the program.

Later on (section D.1 and section D.2 of appendix D) we reproduce almost in full the articles and explanatory memorandums of both draft laws. The explanatory memorandums are particularly interesting, as they highlight the problems currently threatening the public administrations regarding the use of software in general.

6.3.2. Draft law of Brazil

In 1999, parliament member Walter Pinheiro presented a draft law on free software to the Federal Chamber of Brazil (Proposição pl-2269/1999. Dispõe sobre a utilização de programas abertos pelos entes de direito público e de direito privado sob controle acionário da administração pública, Chamber of Deputies of Brazil, December 1999) [185]. This project concerned the use of free software in the public administration and in private companies with the State as majority shareholder.

It recommends the use of free software by these bodies with no restrictions in terms of lending, modification or distribution. The articles of the law describe in detail how free software is defined and how the licences that come with it should be. The definitions coincide with the classical definition of free software by the GNU project. The explanatory memorandum reviews the history of the GNU project, analysing its advantages and achievements. It also refers to the current situation of free software, using the GNU/Linux operating system as an example.

One of the most interesting parts of article one, defines very clearly the sphere in which the use of free software is proposed (bearing in mind that the definition provided in later articles for "open program" is, as already mentioned, the same as free software):

> "The Public Administration at all levels, the powers of the Republic, State and mixed public-private enterprises, public companies and all other public or private bodies subject to the control of the Brazilian State are obliged to use preferably, in their computer systems and equipment, open programs, free of proprietary restrictions with regards to their cession, modification and distribution."

6.3.3.   Draft laws in Peru

In Peru, several draft projects have been proposed on the use of free software by the public administration ("GNU Perú. Draft laws on free software in the public administration of the Peruvian government", Congress of the Republic) [184]. The first and most renowned was proposed by congressman Edgar Villanueva Núñez in December 2001 (Draft law on free software number 1609, December 2001) [222]. It defines *free software* according to the classical definition of the four freedoms (adding perhaps more legal precision, with a definition that specifies six characteristics to be a free program) and proposes its exclusive use in the Peruvian administration:

> "Article 2. The executive, legislative and judicial authorities, decentralised bodies and companies where the State is the majority shareholder, will use exclusively free programs or software in their computer systems and equipment."

Nevertheless, later on, articles 4 and 5 include certain exceptions to this rule.

In its day this draft law had a global repercussion. On the one hand, it was the first time that an administration's exclusive use of free software had been proposed. But even more importantly for the repercussion of this novelty, was the epistolary exchange between congressman Villanueva and Microsoft's representation in Peru, which made allegations against the proposal. This draft law is also interesting in relation to the position adopted by the US embassy, which even sent through official channels a notification (attaching a report prepared by Microsoft) to the Peruvian Congress expressing its "concern over recent proposals by the Congress of the Republic to restrict purchases of the Peruvian Government to open source software or free software " ("Letter to the president of the Congress of the Republic", 2002) [147]. Among other motives, the allegations of both Microsoft and the US Embassy tried to prove that the

draft law would discriminate between different companies making impossible the investments required in order to generate a national industry of software creation. Villanueva argued back that the draft law did not discriminate or favour any particular company in any way, since it did not specify who the Administrator's supplier could be, but rather how (in what conditions) the software would have to be provided. This reasoning is very clear for understanding how the Administration's promotion of free software does not in any way prejudice free competition between providers.

Later on, Peruvian congressmen Edgar Villanueva Núñez and Jacques Rodrich Ackerman presented a new draft law, number 2485, of 8th April 2002 (Draft Law on the Use of Free Software in the Public Administration number 2485, 2002) [223], which in August 2003 was still in parliamentary proceedings. This draft law was an evolution of Draft Law 1609 [222], from which it draws several comments making several improvements, and may be considered a good example of a draft law that proposes the exclusive use of free software in the public administrations, save for certain exceptional cases. Given its relevance, we include its full text (section D.3 of appendix D). In particular, its explanatory memorandum is a good summary of the characteristics that the software used by the public administrations should have and how free software complies with these characteristics better than proprietary software.

6.3.4. Draft laws in Spain

In Spain there have been several legislative initiatives related to free software. Below, we cite a few of them:

- Decree of Measures to Promote the Knowledge Society in Andalucia
  One of the most important legislative initiatives in Spain (because it has come into force) has been unquestionably the one adopted by Andalucia. The Decree of Measures to Promote the Knowledge Society in Andalucia (Decree 72/2003, of 18th March of Measures to Promote the Knowledge Society in Andalucia, 2003) [99] deals with the use of free software, fundamentally (but not only) in the educational context.
  Among others, it promotes the preferable use of free software in public educational centres, obliging all of the equipment purchased by these centres to be compatible with free operating systems, and likewise for the Regional Government centres that provide public Internet access.

- Draft law on Free Software in the context of the Public Administration of Catalonia
  Other communities have debated more ambitious proposals, but without obtaining the majority vote that they required. The most renowned of them is probably the one debated in the Parliament of Catalonia (Proposició de llei de programari lliure en el marc de l'Administració pública de Catalunya, 2002) [221], very similar to the one that the same party (Es-

querra Republicana de Catalunya) presented to the Congress of Deputies, which we will talk about next. This proposal was unsuccessful when submitted for voting.

• Draft Law of Puigcercós Boixassa in the Congress of Deputies

There was also an initiative in the Congress of Deputies proposed by Joan Puigcercós Boixassa (Esquerra Republicana de Catalunya) (Draft Law of Measures for the Implantation of Free Software in the State Administration, 2002) [188]. This initiative proposed the preferable use of free software by the State Administration, and in this sense is similar to other initiatives that share this objective. However, it has the interesting peculiarity of emphasising the availability of localised free programs for the co-official languages (in the autonomous communities that have them). The initiative was not approved in parliamentary proceedings.

7. Free software engineering

"The best way to have a good idea is to have many of them."

Linus Pauling

In previous chapters we have shown why free software's challenge is not the one of a competitor that generates software more quickly, more cheaply and of better quality: free software is different from "traditional" software in more fundamental aspects, starting with philosophical reasons and motivations, continuing with new market and economic models, and ending with a different way of generating software. Software engineering cannot be unaffected by all of the aforementioned factors; so, for a little more than over five years research on how free software is developed has been targeted in greater depth. This chapter aims to discuss the most significant studies and the evidence that they provide, with a view to offering the reader a vision of the state of the art and the future prospects of what we have decided to call *free software engineering*.

7.1. Introduction

Although free software has been developed for several decades now, it is only in recent years that we have started to pay attention to its development models and processes from a software engineering perspective. In the same way as there is no single model for proprietary software development, there is no single model for free software[5] development, but even so we can find interesting characteristics that most of the projects under study share and that could stem from the properties of free programs.

> [5]The article "The ecology of open source software development" (Kieran Healy and Alan Schussman, 2003) [140] shows the large variety of projects and their diversity in numbers of developers, use of tools and downloads.

In 1997, Eric S. Raymond published the first broadly disseminated article *The cathedral and the bazaar. Musings on Linux and open source by an accidental revolutionary*, O'Reilly & Associates http://www.ora.com, 2001) [192], describing some of the characteristics of free software development models, making special emphasis on what distinguished these models from those of proprietary development. Since then, this article has become one of the most renowned (and criticised) in the world of free software, and up to a point, the sign of the starting development of free software engineering.

7.2. The cathedral and the bazaar

Raymond makes an analogy between the way of building mediaeval cathedrals and the classical way of producing software. Arguing that in both cases there is a clear distribution of tasks and functions, emphasising the existence of a designer who oversees everything and has to control the development of the

activity at all times. At the same time, planning is strictly controlled, giving rise to detailed processes where ideally each participant in the activity has a clearly defined role.

What Raymond takes as the model for building cathedrals not only has room for the heavy processes that we can find in the software industry (the classical waterfall model, the different aspects of the Rational Unified Process, etc.), but also for free software projects such as GNU and NetBSD. For Raymond, these projects are highly centralised, since just a few people are responsible for the software's design and implementation. The tasks carried out by these people, in addition to their functions, are well defined, and anyone wishing to form part of the development team needs to be assigned a task and a function according to the project's requirements. On the other hand, releases of this type of programs are spaced in time according to a fairly strict schedule. This means having few software releases and long cycles, consisting of several stages between releases.

The opposite model to the cathedral is that of the bazaar. According to Raymond, some of the free software programs, particularly the Linux kernel, have been developed following a similar scheme to that of an oriental bazaar. In a bazaar there is no maximum authority to control the processes that are developed or to strictly plan what has to happen. At the same time, participants' roles can change continuously (sellers can become clients) and with no outward indication.

But what is most novel about "The cathedral and the bazaar" is how it describes the process by which Linux has become a success; it is a list of "good practices" to make the most of the opportunities offered by the source code being available, and of interactivity through the use of telematic systems and tools.

A free software project tends to appear as a result of a purely personal action; the cause can be found in a developer who finds his ability to resolve a problem limited. The developer needs to have enough knowledge to start solving it, at least. Once he has obtained something usable, with some functionality, simple, and if possible, well designed or written, the best he can do is to share that solution with the world of free software. This is what is known as *release early*, which helps to draw the attention of other people (usually developers) who have the same problem and who may be interested in the solution.

One of the basic principles of this development model is to think of users as co-developers. They need to be treated with care, not only because they can provide "word of mouth" publicity but also because they will carry out one of the most costly tasks that there is in software generation: testing. Unlike co-development, which is difficult to scale, debugging and tests have the property of being highly parallelisable. The user will be the one to take the software

and to test it on his machine under specific conditions (an architecture, certain tools, etc.), a task that multiplied by a large number of architectures and environments would entail an enormous effort for the development team.

If we treat users as co-developers it could happen that one of them finds a bug and resolves it by sending a patch to the project developers so that the problem can be solved in the following version. It can also happen, for example, that someone other than the person who discovers the bug eventually understands it and corrects it. In any case, all of these circumstances are beneficial for the development of free software, i.e. it is beneficial to enter a dynamic of this type.

This situation becomes more effective with frequent releases, since the motivation to find, notify and correct bugs is high because it is assumed that they will be attended immediately. Also, secondary benefits are achieved such as the fact that frequent integration - ideally once or more times a day - does not require a final phase of integrating the modules comprising the program. This has been called *release often* and allows a great modularity (Alessandro Narduzzo and Alessandro Rossi, "Modularity in action: GNU/Linux and free/open source software development model unleashed", May 2003) [176], at the same time as it maximises the propaganda effect provided by the publication of the software's latest version.

Note

New version management depends, logically, on the size of the project, since the problems that need to be dealt with are not the same when the development team has two members as when it has hundreds. Whereas, in general, for small projects this process is more or less informal, the management of releases for large projects tends to follow a defined process, which is not exempt from a certain degree of complexity. There is an article called "Release management within open source projects" (Justin R. Ehrenkrantz, 2003) [110] which describes in detail the sequence followed with the Apache web server, the Linux kernel and the Subversion versioning system.

In order to prevent "release often" from frightening users with a priority for the stability of the software over the speed with which the software evolves, some free software projects have several development branches running in parallel. The most renowned case of this is the Linux kernel, which has a stable branch directed at those who value its reliability and another unstable one designed for developers with the latest innovations and novelties.

7.3. Leadership and decision-making in the bazaar

Raymond suggests that all free software projects should have a *benevolent dictator*, a sort of leader who is normally the founder of the project to guide the project and always have the last word when it comes to decision-making. The skills that this person must have involve mainly knowing how to motivate and coordinate a project, understanding users and co-developers, seeking con-

sensus and integrating everyone who has something to contribute. As you can see, we have not mentioned technical competence among the most important requirements, although it is never superfluous.

As the size of projects and the number of developers involved with them have grown, new ways of organising decision-making have emerged. Linux, for example, has a hierarchical structure based on Linus Torvalds delegating responsibilities, the "benevolent dictator". And, we will see that there are parts of Linux that have their own "benevolent dictators", although their power will be limited by the fact that Linus Torvalds has the last word. This case is a clear example of how a high level of modularity in a free software project has given rise to a specific way of organising things and making decisions (Alessandro Narduzzo and Alessandro Rossi, "Modularity in action: GNU/Linux and free/open source software development model unleashed", 2003) [176].

Note

Some people claim that the way free software projects are organised is similar to a surgical team, as proposed by Harlan Mills (of IBM) in the early seventies popularised by Brooks in his famous book *The mythical man-month* (Frederick P. Brooks Jr., 1975) [150]. Although there may be cases where the development team of a particular free software application consists of a designer/developer (the surgeon) and many co-developers who perform auxiliary tasks (systems administration, maintenance, specialised tasks, documentation) there is never such a strict and defined separation as the one suggested by Mills and Brooks. All in all, as Brooks points out in the case of the surgical team, in free software the number of developers that need to communicate in order to create a big and complex system - the most active ones - is much lower than the total number of developers.

In the case of the Apache Foundation, we have a *meritocracy*, since this institution has a directors' committee consisting of people who have contributed in a notable way to the project. In reality, it is not a strict meritocracy in the sense of those who most contribute govern, since the directors' committee is elected democratically and regularly by the Foundation's members (responsible for managing various free software projects, like Apache, Jakarta, etc.). To become a member of the Apache Foundation, you need to have contributed in an important and continuous way to one or several of the Foundation's projects. This system is also employed by other large projects, such as FreeBSD or GNOME.

Another interesting case of formal organisation is the GCC Steering Committee. It was created in 1998 to avoid anyone obtaining control over the GCC project (GNU Compiler Collection, GNU's compiler system) and backed by the FSF (promoter of the GNU project) a few months later. In a certain sense, this committee continues the tradition of a similar one that the EGCS project had (which for a time ran in parallel to the GCC project, but later joined it). Its fundamental mission is to ensure that the GCC project fulfils the project's *mission statement*. The committee's members are members in a private capacity, and are selected by the project itself in such a way as to faithfully represent

the different communities that collaborate in the GCC's development (support developers for several programming languages, developers related to the kernel, groups interested in embedded programming, etc.).

The same person does not have to be the leader of a free software project forever. Basically, there can be two circumstances in which the project leader stops being so. The first is lack of interest, time or motivation to continue. In this case, the baton must be passed to another developer who will assume the role of project leader. Recent studies (Jesús M. González Barahona and Gregorio Robles, 2003) [87] show that, in general, project leadership frequently changes hands, in such a way that we can see several *generations* of developers over time. The second case is more problematic: it involves a forking. Free software licences allow code to be taken, modified and redistributed by anybody without requiring the project leader's approval. This does not normally tend to happen, except in cases where the idea is to deliberately avoid the project leader (and the leader's potential veto against a contribution). This is similar on the one hand to a sort of "coup d´etat", which on the other hand is totally licit and legitimate. For this reason, one of a project leader's objectives in keeping co-developers satisfied is to minimise the possibility of a forking.

7.4. Free software processes

Although free software is not necessarily associated with a specific software development process, there is a broad consensus about the processes that it most commonly uses. This does not mean that no free software projects have been created using classical processes, such as the waterfall model. In general, the development model of free software projects tends to be more informal, due mostly to the fact that a large share of the development team performs these tasks voluntarily and in exchange for no financial reward, at least directly.

The way of capturing requirements in the world of free software depends as much on the "age" as on the size of the project. In the early stages, the project's founder and the user tend to be the same person. Later on, and if the project expands, the capture of requirements tends to take place through electronic mailing lists and a clear distinction tends to be reached between the development team, or at least, the more active developers and the users. For large projects, with many users and many developers, requirements are captured using the same tool as the one used for managing the project's bugs. In this case, instead of dealing with bugs, they refer to activities, although the mechanism used for managing them is identical to the one for debugging (they will be classified in order of importance, dependency, etc., and it will be possible to monitor whether they have been resolved or not). The use of this planning tool is fairly recent, so we can see how the world of free software has evolved somewhat from a total lack, to a centralised system for managing these activ-

ities in engineering terms, even if it is certainly more limited. All in all, it is not usual to find a document that gathers the requirements, as is normally the case in the waterfall model.

As for the system's global design, only large projects tend to have it documented in comprehensive detail. For the rest, the main developer (or group of main developers) is most likely the only one to have it, in their head; sometimes, this is even not the case, and the system takes shape as the software evolves. The lack of a detailed design not only imposes limitations regarding the possible reuse of modules, but also is a large obstacle when it comes to giving new developers access, since they will have to face a costly and slow learning process. Having a detailed design is not very common either. The lack of it means that many opportunities for reusing code are lost.

Implementation is the phase where free software developers concentrate most effort, among other reasons because in their view it is clearly the most fun. To do this, the classical programming model of trial and error is normally observed until the desired results are achieved from the programmer's subjective point of view. Historically, it is rare that unit tests are included with the code, even if they would make modification or inclusion of subsequent code by other developers easier. In the case of certain large projects, such as Mozilla for example, there are machines exclusively dedicated to downloading repositories containing the most recent code and to compile it for different architectures ("An overview of the software engineering process and tools in the Mozilla project", 2002) [193]. Detected bugs are notified to a mailing list of developers.

However, automatic tests are not an entrenched practice. In general, users themselves, with their enormous range of uses, architectures and combinations, will carry them out. This has the advantage of running them in parallel at a minimum cost for the development team. The problem with this model is how to obtain feedback from users and organise it as efficiently as possible.

As far as software maintenance in the world of free software is concerned, understood as the maintenance of previous versions, having this task will depend on the project. For projects that need stability, such as operating system kernels, previous versions are maintained, since changing to a new version can be traumatic. But in general, for most free software projects, if a bug is found in a previous version, developers will usually ignore it and recommend the use of the latest version in the hope that the bug has disappeared with the software's evolution.

7.5. Criticism of "The cathedral and the bazaar"

"The cathedral and the bazaar" suffers from not being systematic and a lack of rigour given its journalistic rather than scientific nature. The most frequent criticisms refer to the fact that it basically explains the particular case of the Linux experience and aims to extend those conclusions to all free software

projects. In this sense, in "Cave or community? An empirical examination of 100 mature open source projects" [160] we can see that the existence of a community as large as the community of the Linux kernel is an exception rather than the rule.

Even more critical are those who believe that Linux is an example of the cathedral development model. They argue that obviously there is a driving force, or at least a person with maximum authority, and a hierarchical system that delegates responsibility down to the labourers/programmers. Also, there is a distribution of tasks, albeit implicitly. "A second look at the cathedral and the bazaar" [91] goes beyond and maintains, not without a certain level of bitterness and arrogance in its reasoning, that the metaphor of the bazaar is internally contradictory.

Another of the most criticised points of "The cathedral and the bazaar" is its assertion that the Brooks law, which states that "adding developers to a delayed software project delays it even more" (*The mythical man-month. Essays on software engineering*, 1975) [150], is not valid in the world of free software. In [148] we can read how what happens in reality is that the environmental contexts are different and that what in principle appears to be incongruent with Brooks' law, after a more comprehensive analysis, is just a mirage.

7.6. Quantitative studies

Free software makes it possible to go deeper into the study of code and other parameters that intervene in its generation thanks to having access to many public information sources. This allows areas of traditional software engineering such as empirical software engineering to be fostered due to the existence of a huge amount of information that can be accessed without the need to heavily intrude in the development of free software. The authors are convinced that this vision can contribute enormously to the analysis and comprehension of the phenomena associated with free software (and software in general), and that it may even, among other possibilities, manage to produce predictive software models with feedback in real time.

The idea behind it is very simple: "given that we have the opportunity to study an immense number of free software programs, let's do so." And in addition to a project's present status, its past evolution is public, meaning that all of this information, duly extracted, analysed and packaged, can serve as a knowledge base that allows us to evaluate a project's state of *health*, helping towards decision-making and foreseeing current and future complications.

The first quantitative study of any importance in the world of free software dates back to 1998, although it was published in early 2000 ("The Orbiten free software survey") [127]. Its purpose was to find out in empirical terms the participation of developers in free software. To do so they statistically processed the authorship assignments that authors tend to place

in the heading of source code files. The results showed that participation was consistent with the Pareto law ("Course of Political Economy", Lausana, 1896) [182]: 80% of the code corresponds to the most active 20% of developers, whereas the remaining 80% of developers contribute 20% of the total code. Many subsequent studies have confirmed and extended the validity of this result to different ways of participating in the contribution of source code (mailing lists, bug notifications or even the number of downloads, as we can see in http://www-mmd.eng.cam.ac.uk/people/fhh10/Sourceforge/Sourceforge%20paper.pdf [145]).

Note

The fact that many economic terms appear in the study of free software engineering is a result of the interest some economists have shown in learning about and understanding what motivates volunteers to produce high value goods without usually obtaining a direct benefit in exchange. The most well-known article is "Cooking pot markets: an economic model for the trade in free goods and services on the Internet" [125], which introduces the idea of the *gift economy* on Internet. At http://www.wikipedia.org/wiki/Pareto [232] we can obtain further details on the Pareto principle and its generalisation to the Pareto distribution. The Lorenz curve (http://www.wikipedia.org/wiki/Lorenz_curve) [231], which graphically shows developers' participation in a project, is also interesting as well as the Gini coefficient (http://www.wikipedia.org/wiki/Gini_coefficient) [230], calculated on the basis of the Lorenz curve and which produces a number that shows the system's inequality.

The tool used to conduct this study was published by its authors under a free licence, meaning that its results can be reproduced and it can be used to conduct new studies.

In a later study, Koch ("Results from software engineering research into open source development projects using public data", 2000) [158] went further and also analysed the interactions in a free software project. The information sources were mailing lists and the repository of versions of the GNOME project. But the most interesting aspect of the Koch study was the economic analysis. Koch focuses on checking the validity of classical cost forecasts (function points, COCOMO model...) and shows the problems involved in applying them, although it does admit that the results obtained have to be taken with due reserve do partly match reality. He concludes that free software requires its own models and methods of study, since known ones are not adapted to its nature. However, obviously being able to obtain much of the data related to the development of free software publicly, allows us to be optimistic about achieving these objectives in the near future. Koch's can be considered the first full quantitative analysis, although it certainly lacks a clear methodology, and especially some *ad hoc* tools that would have made it possible to verify its results and to study other projects.

In the year 2000, Mockus *et al.* presented the first study of free software projects encompassing a full description of the development process and organisational structures, with both qualitative and quantitative evidence ("A case study of open source software development: the Apache server") [172]. To do so, they used the software changelog and bug reports to quantify aspects of developers' participation, kernel size, code authorship, productivity, fault

density, and problem-solving intervals. In a way, this study is still a classical software engineering study, save for the fact that the data has been integrally obtained from the semi-automatic inspection of the data that the projects offer publicly on the net. As in the case of "Results from software engineering research into open source development projects using public data", 2000 [158], this article did not provide any tool or automatic process that could be reused in future by other research teams.

In "Estimating Linux's size", 2000 [227], and "More than a gigabuck: estimating GNU/Linux's" [228] we find a quantitative analysis of the lines of code and programming languages used in the Red Hat distribution. González Barahona *et al.* have followed these steps in a series of articles on the Debian distribution (*vid.* for example "Anatomy of two GNU/Linux distributions" [88]). All of these provide a sort of *X-ray* of these GNU/Linux distributions on the basis of data provided by a tool that counts a program's source lines of code (SLOC, lines of code that are not blank lines or comments). Aside from the spectacular result in total lines of code (Debian 3.0 known as Woody, has more than one hundred million lines of code), we can see how the number of lines is distributed for each programming language. Being able to study the evolution of the different Debian versions over time has thrown up some interesting results [88]. It is worth noting that in the last five years the average package size has remained practically constant, meaning that the natural tendency to grow has been neutralised by the inclusion of smaller packages. At the same time, we can see how the importance of the C programming language, though still predominant, is declining over time, whereas script languages (Python, PHP and Perl) and Java are experiencing an explosive growth. The "classical" compiled languages (Pascal, Ada, Modula...) are clearly receding. Finally, these articles include a section that shows the results obtained if we apply the classical COCOMO effort estimate model dating from the early eighties (*Software Engineering Economics*, 1981) [93] and which is used by proprietary software to estimate effort, project schedules and costs.

Although precursors, most of the studies presented in this section are fairly limited to the projects under analysis. The methodology employed has been adapted to the analysed project, is partly manual and occasionally the automated part can be used generally with other free software projects. This means that the effort required to study a new project is much greater, since the method needs to be readapted and the manual tasks will have to be repeated.

For this reason, the latest efforts ("Studying the evolution of libre software projects using publicly available data", in: *Proceedings* of the 3rd Workshop on Open Source Software Engineering, 25th International Conference on Software Engineering, Portland, USA [196] or "Automating the measurement of open source projects", 2003 [124]) focus on creating an analysis infrastructure that integrates several tools so that the process can be automated to a maximum. There are two fairly obvious reasons for doing this: the first is that once a lot

of time and effort has been invested in creating a tool to analyse a project with special emphasis on making it generic, the effort involved in using it for other free software projects is minimal. The second is that analysis using a series of tools that study programs from different and sometimes complementary points of view, at times does not allow us to obtain a broader vision of the project. In the Libre Software Engineering Web Site [86] we can follow these initiatives in more detail.

7.7. Future work

Having described the brief but intense history of software engineering research on free software, we can say that it is still taking its first steps. Many important aspects are still pending analysis and detailed examination until we can find a model that at least partly explains how free software is generated. The issues that will need to be tackled in the near future include the classification of free software projects, the creation of a methodology based inasmuch as possible on automated analysis and the use of acquired knowledge to build models that help us to understand how free software develops at the same time as facilitating decision-making on the basis of acquired experience.

Another aspect that should not be overlooked and that is starting to be considered now is the validity of classical engineering methods in the field of free software across all software engineering intensifications. Hence, for example, the laws of software evolution postulated by Lehman ("Metrics and laws of software evolution - the nineties view" [165]) at the beginning of the nineteen seventies and updated and expanded in the eighties and nineties appear not to be fulfilled unconditionally in the development of some free software projects ("Understanding open source software evolution: applying, breaking and rethinking the laws of software evolution", 2003 [199]).

Currently, one of the most serious deficiencies is the lack of a strict classification so that free software projects can be classed into different categories. At present, the classification criteria are too broad, and projects with very disparate organisational, technical or other characteristics are all put into the same bag. The argument that Linux, with an extensive community and large number of developers, has a different nature and does not behave in the same way as a much more limited project in numbers of developers and users, is very true. All in all, a more detailed classification would make it possible to reuse the experience acquired in other similar projects (in other words, with similar characteristics), making it easier to make forecasts, and making it possible to foresee risks, etc.

The second important aspect that free software engineering needs to tackle, closely connected to the preceding point and current trends, is the creation of a methodology and tools to support it. A clear and concise methodology will make it possible to study all projects on an equal footing, discover their current status, learn how they have evolved, and of course, classify them. Tools are

essential when it comes to dealing with this problem, since once created they make it possible to analyse thousands of projects with minimum additional effort. One of the objectives of free software engineering is to make it possible to study a project in depth on the basis of a limited set of parameters showing where information on the project can be found on the Net (the address of the software versions repository, the place where the mailing list archives are stored, the location of the bug management system, and a minimum survey). Project managers would then be just a button away from a complete analysis, a sort of *clinical analysis* that helped to diagnose a project's state of *health* including at the same time indications on areas for improvement.

Once we have acquired methods, a classification and models, the opportunities arising from simulation, and to be more precise, intelligent agents, could be enormous. Considering that our starting point is a notoriously complex system, it would be interesting to create dynamic models on which the different entities participating in software generation could be modelled. Obviously, the more we know about the different elements, the more adapted to reality our model will be. Although several proposals for free software simulation are known, they are fairly simple and incomplete. To some extent, this is due to the fact that there is still an enormous lack of knowledge with regards to the interactions that take place in the generation of free software. If we manage to correctly package and process projects' information throughout their history, the agents could become crucial for knowing what their future evolution will be. Although there are many proposals as to how to approach this problem, one of the most advanced for now can be found at http://wwwai.wu-wien.ac.at/~koch/oss-book/ [82].

7.8. Summary

In summary, we have tried to show in this chapter that free software engineering is still a young and unexplored field. Its first steps are due to journalistic essays that proposed, not without a certain lack of scientific rigour, a more efficient development model, but gradually progress has been made towards a systematic study of free software from an engineering perspective. Currently, following several years of reports and quantitative and qualitative analysis of free projects, an enormous effort is being made to achieve a global infrastructure that makes it possible to classify, analyse and model the project within a limited space of time and in a partly automated manner. When analysing free software projects stops being so costly in time and effort as it is now, it is likely that a new stage in software engineering will begin, with a different type of techniques appearing on the scene designed mainly to predict software evolution and foresee potential complications.

8. Development environments and technologies

"The tools we use have a profound (and devious!) influence on our thinking habits, and, therefore, on our thinking abilities."

Edsger W. Dijkstra, "How do we tell truths that might hurt?"

Down the years free software projects have created their own (also free) tools and systems to contribute to the development process. Although each project follows its own rules and uses its own set of tools, there are certain practices, environments and technologies that can be considered usual in the world of free software development. In this chapter we will look at the most common ones and discuss their impact on projects' management and evolution.

8.1. Description of environments, tools and systems

Before explaining about specific tools, we will define their general characteristics and properties according to the task to be performed and the way developers are organised.

Firstly, although it is not necessarily a determining factor, it is common for the environment, development tools (and even the target virtual machine, when there is one), also to be *free*. This has not always been the case. For example, the GNU project, with the objective of replacing Unix, had to be developed in and for proprietary Unix systems until Linux and the free BSDs appeared. Nowadays, especially when free software is developed as part of a business model, the tendency is that the target machine can also be a proprietary system, often through interposed virtual machines (Java, Python, PHP, etc.). In any case, the environment and the virtual machine need to be sufficiently common and cheap to bring together enough co-developers having the same tools.

Secondly, also in order to attract the largest possible number of co-developers, the tools need to be *simple*, well *known* and capable of functioning on *economical* machines. Perhaps for these reasons the world of free software is fairly conservative when it comes to languages, tools and environments.

In the third place, the free software development model tends to be eminently distributed, with many potential collaborators spread all around the world. For this reason generally asynchronous collaboration tools are necessary, which at the same time allow the development to progress easily, irrespective of the amount and rhythm of work of each collaborator, without delaying anyone.

Finally, it is advisable to provide developers with various different architectures on which they can compile and test their programs.

8.2. Associated languages and tools

Most free software is written in C language, not only because C is the natural language of any Unix variant (the usual free software platform), but also because it is widespread, both in people's minds and in the machines (GCC is a standard compiler installed by default in almost every distribution). Precisely for these reasons and for its efficiency, Stallman recommends its use in GNU projects ("GNU coding standards") [203]. Other fairly similar languages are C++, also supported by default by GCC, and Java, which has certain similarity and is popular because it allows developments for virtual machines available in a wide range of platforms. Generally, software engineering reasons are not taken into account: in SourceForge (*vid.* section 8.9.1), in 2004, for every one hundred and sixty projects in C there was one in Ada, although the latter is supposedly a more appropriate language for developing quality programs. At the same time, English is the *lingua franca* of free software developers, despite the fact that Esperanto is a much easier language to learn with a much more logical structure. Interpreted languages designed for the rapid prototyping of normal applications and web services such as Perl, Python and PHP are also popular.

Just as C is the standard language, *make* is the standard program building tool, given its source code files. A free programmer will normally use the GNU version (GNU make) [36] rather than BSD's incompatible one (Adam de Boor, "PMake - a tutorial") [100]. They can be used to specify dependency trees between files, and rules for generating dependent files from those that they depend on. Thus, we can specify that an object file `x.o` depends on source files `x.c` and `x.h` and that to build it we need to execute **`gcc -c x.c`**. Or that our program's executable depends on a collection of objects and is linked in a certain way. When we modify source code and then execute *make*, only the affected modules will be recompiled and the final object will be linked again. This is a very low level tool, since, for example, it is incapable of finding out for itself when a module needs to be recompiled in C, despite the fact that it could do so by examining the chains of *includes*. It is also very powerful, because it can combine all the file transformation tools available in order to build very complex targets of a multi-language project. But it is very complicated and very dependent on Unix-type environments. Other supposedly better alternatives, such as *jam* (Jam Product Information) [41], *aap* (Aap Project) [1] or *ant* (The Apache Ant Project) [7] are rarely used (the latter is gaining popularity especially in the world of Java).

Given the heterogeneity of existing systems even in the world of Unix, we also use tools designed to help make our programs portable. The GNU tools *autoconf* (http://www.gnu.org/software/autoconf) [10], *automake* (http://www.gnu.org/software/automake) [32] and *libtool* (http://www.gnu.org/software/libtool) [35] make these tasks easier in C and Unix environments. Given the diversity of languages, character sets and cultural contexts, C programmers (and programmers using many other languages) often use *gettext* (http://

www.gnu.org/software/gettext) [31] and the internationalisation options of the standard C library (http://www.gnu.org/software/libc) [34] for programming applications that can be easily localised to any cultural environment at runtime.

Thus, when we receive a source package, it is most likely written in C, packaged with *tar*, compressed with *gzip*, made portable with *autoconf* and associated tools, and can be built and installed with *make*. Its installation will be carried out in a very similar process to the following one:

```
tar xzvf package-1.3.5.tar.gz cd package-1.3.5./configure make make install
```

8.3. Integrated development environments

An IDE (*integrated development environment*) is a system that makes software developer's work easier by solidly integrating the language oriented edition, the compilation or interpretation, debugging, performance measurements, incorporation of source code to a source control system, etc., normally in a modular fashion.

Not all free software developers like these tools, although their use has gradually expanded. In the world of free software, the first one to be extensively used was GNU Emacs (http://www.gnu.org/software/emacs/) [33], star work of Richard Stallman, written and extensible in Emacs Lisp, for which there are lots of contributions.

Eclipse (Eclipse - An Open Development Platform) [23] can be considered today's reference IDE in the world of free software, with the disadvantage that it works better (around May 2007) on a non-free virtual Java machine (Sun's which is hoped to become free soon anyway). Other popular environments are Kdevelop (http://www.kdevelop.org) [42] for KDE, Anjuta (http://www.anjuta.org) [6] for GNOME, Netbeans (http://www.netbeans.org) [51] of Sun for Java and Code::Blocks (http://www.codeblocks.org) [18] for C++ applications.

8.4. Basic collaboration mechanisms

Free software is a phenomenon made possible by the collaboration of distributed communities and that, therefore, requires tools to make that collaboration effective. Although for a long time magnetic tapes were physically posted, the speedy development of free software began once it became possible to communicate rapidly with many people and to distribute program code to them or reply with comments and patches. For convenience, rather than

sending code, messages could be used to send information on the site from which the code could be collected. In fact, right in the beginning of the seventies, e-mail was an extension of the ARPANET file transfer protocol.

In the world of Unix, in the mid-seventies, *uucp*, the Unix file transfer protocol, was developed for communicating machines through dial-up and dedicated lines, and on which electronic mail was built, and in 1979, the first USENET link over UUCP. USENET *news*, a hierarchically structured forum system distributed by flooding to hierarchically arranged sites, played a fundamental role in the development of free software, sending the source code of complete programs to the `comp.sources` groups.

Simultaneously, mailing lists were developed, among which the BITNET (1981) mailing list managers deserve mention. Nowadays the tendency is to prefer mailing lists over USENET-type newsgroups. The main reason has been the abuse for commercial purposes and intrusion of "absentminded" people, interfering with noise in the discussions. Also, mailing lists provide more control and can reach more people. Recipients need to subscribe and any e-mail address is valid, even if there is no direct Internet access. The mailing list administrator can choose to know who subscribes or to unsubscribe someone. The contributions can be restricted to members only or the programmer may choose to moderate the articles before they appear[6].

> [6]There are also moderated newsgroups

Traditionally, mailing list administration has been done by e-mail, using special messages with a password, allowing the administrator not to have permanent Internet access, although this is becoming an increasingly rare phenomenon, meaning that the most popular mailing lists manager nowadays (Mailman, the GNU Mailing List Manager) [46] cannot be administrated by e-mail, but rather necessarily via the web. The mailing lists play a crucial role in the world of free software and in many cases[7] they may be the only way to contribute.

Currently, with the web's popularity, many forums are pure web forums or *weblogs*, with no other interface than the one provided by the navigator. They can be generic, like the popular SlashDot (Slashdot: News for Nerds") [58] or the spanish Barrapunto (http://barrapunto.com) [11], where new free software is announced or discussed. Or they can be specialised in a specific program; in this case they are often integrated with several additional tools in collaboration sites (*see* section 8.6.2). There are also web interfaces to newsgroups and traditional lists.

> [7]For example, contributions to Linux have to be made as text patches to the list `linux-kernel@vger.kernel.org`.

Another collaboration mechanism that has become popular at the same time, is based on *wikis*, especially when the idea is to build a joint document, such as the specification for a program, a module or a system. We discuss this in section 8.6.2.

Finally, we should mention the interaction mechanisms used by developers to converse in real time. For free software it does not tend to be a practical mechanism, because with all the developers distributed around the world it is not easy to find a convenient time for everyone. Nonetheless, there are several projects that use these text chat tools, either regularly or at virtual conferences on set dates. The most commonly used tool is the IRC (Internet Relay Chat, http://www.ietf.org/rfc/rfc2810.txt) [151], which normally communicates people through themed "channels" established on the basis of a series of collaborating servers. It is not common for multimedia tools to be used (sound, image.,) probably because quality connections are required which not everyone may have and that can entail problems with the free software available, and the difficulty of registering and editing the results of conversations for documenting purposes.

8.5. Source management

It is advisable for any program development project to archive its history, because a modification could produce a hidden error discovered later for example, and the original needs to be recovered, at least in order to analyse the problem. If the project is developed by several people, the author of each change will also need to be recorded, for the same reasons as explained above. If versioned releases of a project are made, we need to know exactly which versions of each module form part of each release. Often, a project will keep one stable version and another experimental version; both need to be maintained, debugged, and corrected errors transferred from one version to the other. This can all be done by saving and labelling each and every version of the files correctly, which has generally been considered an excessive cost, although with current drives this is becoming less true. What a *source control system* , also known as a *version management system* normally does, is to save the file history as a set of differences against a version, normally the most recent one, for efficiency, also labelling each difference with the necessary metadata.

But we also want a system of these characteristics to serve for many programmers to collaborate effectively without stepping on each other's toes, but without impeding each other's progress. Therefore, we need to be able to allow several programmers to work concurrently, but with a certain level of control. This control can be optimistic or pessimistic. With pessimistic control, a programmer can reserve some files to himself to improve for a time, during which nobody else can touch those files. This is very safe, but will block other programmers and may delay the project, especially if the programmer that has locked the files is busy with other things or has even forgotten about them. Allowing others to progress is more dynamic, but more dangerous, since incompatible modifications can occur. An optimistic system allows progress to be made, but warns us when there have been conflicts and gives us tools to resolve them.

8.5.1.   CVS

CVS (Concurrent Version System) is an optimistic source management system designed towards the end of the eighties and used by the vast majority of free projects (Concurrent Version System [20], *Open source code development with CVS*, 2nd edition [113],Version Management with CVS [95]). It uses a central repository accessed through a client/server system. The site administrator decides who has access to the repository, or to which parts of the repository, although normally, once a developer has been admitted within the circle of trust, he will have acess to all files. Anonymous access, in read-only mode, may also be allowed for anyone.

The anonymous collaborator

The *anonymous CVS* is a vital tool for fulfilling the "release early and often" concept advocated by Eric Raymond. Any user anxious to try the latest version of a program can extract it from the CVS, discover bugs and report them, even in the form of patches with the correction. And it can examine the full history of the development.

Let's look a bit at the mechanics. An advanced user wishes to obtain the latest version of the module `mod` from an anonymously accessible repository in `progs.org`, directory `/var/lib/cvs` and protocol `pserver`. The first time he will declare his intention to enter:

```
cvs -d:pserver:anonymous@progs.org:/var/lib/cvs login
```

If a password is requested, it will be anonymous user (usually the carriage return), which will be registered in a local file (this operation is not really necessary for anonymous access, but the program will complain if the file with the password does not exist). Next, the important thing is obtain the first copy of the module:

```
cvs -d:pserver:anonymous@progs.org:/var/lib/cvs co mod
```

This will create a directory `mod` with all of the module's files and directories and some metadata (contents in subdirectories called `CVS`), which will allow, among other things, not having to repeat the information already provided. Our advanced user will enter the created directory, generate the package and test it:

```
cd mod./configure make make install...
```

When he wishes to obtain a new version, he will simply update his copy within `mod`.

```
cd mod cvs update./configure make make install...
```

If he finds a bug, he can correct it in place and then send a patch via e-mail to the program's maintainer (individual or mailing list):

```
cvs diff -ubB | mail -s "My patches" mod-maint@progs.org
```

The normal developer

The normal developer will have an account on the server. He can use the same mechanism and the same protocol as the anonymous user, replacing `anonymous` for his account name.

Once he has a working copy of the module, he can make the necessary changes, and when he considers that they have been stabilised, *commit* the changes to the repository. For example, if he modifies the files `part.h` and `part.c`, he will commit them like this:

Note
For security reasons, for accounts with write permissions, `ssh` tends to be used, as it provides an authenticated and encrypted channel.

```
cvs ci part.h part.c
```

Before completing the operation, the CVS will ask him for an explanation of what he has done, which will be attached to both files' log. Also the *revision number* of each file will be increased by one unit. This number identifies every important moment in the history of a file and can be used to recover each one of those moments.

When should a developer do a commit? This is a question of methodology that project members need to agree, but it seems obvious that changes that do not compile should not be committed. But it is preferable to pass also a minimum test battery. In many projects the approval of a project or sub-project supervisor who examines the modification is also required.

In developing the modification, someone may have altered other files, or even the same ones. Therefore it is advisable for developers to do a relatively frequent update of their copy (*cvs update*). If other files have been modified, the environment may also have changed and tests that were previously passed may now be failed. If the same files have been modified, it could be that these changes have occurred either in places or routines that we have not touched or in code that we have modified. In the first case there is no conflict (at least not apparent) and the modification operation "merges" our version with

the repository's, generating combined files, with all of the changes. Otherwise there is a conflict, in which case we need to discuss with the developer who has made the other changes and agree to a final version.

For better identification of each project component, it is advisable for it to carry directly associated revision information. CVS can label source codes and objects automatically, on condition of following a certain discipline. For example, if in a source code comment we write the key word $Id, every time the file is committed to the repository, the word will be replaced with an identification chain that will show the file name, the revision number, [8]the date and time of the commit and its author:

```
$Id: part.c,v 1.7 2003/07/11 08:20:47 joaquin Exp $
```

If we include this keyword in a string of the program, when compiled the string will appear in the object and in the executable, making it possible to identify it with a tool (*ident*).

The administrator

Obviously, administrators are responsible of the most complicated part of maintaining the repository. For example, they need to register the program, issue permissions for developers and coordinate them, label delivered versions, etc.

It is common practice for all projects to have a stable version and an experimental version. To do this we create branches. Whereas those dedicated to maintenance correct errors on the stable branch, new developments are made on the experimental branch. When the experimental branch stabilises, it is passed onto stable, but not without previously applying the corrections made to the former stable branch. This operation is called *merging*, it is delicate and supported by CVS, although in a somewhat primitive way. This idea can be extended to the concept of experimental branches which evolve in different directions, which may or may not come to a good end, and that in any case, unless they are dead ends, will have to be fully or partly integrated into the stable product, with appropriate merges.

A right that free software gives us is to modify a program for private use. Although it is desirable to contribute all improvements to the common pool, often the modifications we wish to make are too specific and uninteresting for the public at large. But we are interested in incorporating the evolution in the original program. This can be done with a special type of branching and merging (*vendor branches*).

The administrator can also facilitate team coordination through automated mechanisms, such as by generating e-mail messages when certain events occur, like commits, or forcing certain automatic actions to be carried out before a commit, such as automatic checks of style, compilations, or tests.

8.5.2. Other source management systems

Despite being the most extensively used version control system, CVS has some notable disadvantages:

1) CVS does not support either renamings or file directory changes, or metadata (owner, permissions, etc.) or symbolic links.

2) Because it is an evolution of a version control system for individual files, it naturally does not support version control for complete groups.

3) CVS does not support sets of coherent changes. Indeed, adding a feature or correcting an error can involve changing several files. These changes should be atomic.

4) In CVS the use of branches and merges is fairly complicated. In fact, if we create an experimental branch of a project and wish to include the corrections made to the stable version, we need to know in detail which corrections have been made already and which not, so as not to do them several times over.

5) CVS depends on a centralised server, and although it is possible to work without a connection, we do need one for generating versions, comparing and merging them.

6) CVS does not generate, without the help of separate tools, the file `changelog`, which shows the global history of a project's changes.

7) CVS does not support well projects with a very large number of files, as in the case of the Linux kernel.

Note

In 2007 `Subversion` is already the clear successor of CVS, and many free software developments have migrated to it.

And yet, there are other free systems which solve several of these problems. We would highlight the already mentioned successor of CVS, Subversion (http://subversion.tigris.org) [62], (http://svnbook.red-bean.com/) [96], which strictly solves the basic problems of CVS and can use HTTP extensions (WebDAV) in order to bypass aggressive security policies.

The development model based on a centralised repository, although suitable for cooperative work, does not satisfy all expectations, since being able to create our own development branches depends on the one hand on the server's accessibility and good functioning and on the other on the administrators of that server. Sometimes distributed repositories are required that allow anyone

to have a repository with a private or public branch that can be merged or not with the official one. This is how *GNU arch* (Arch Revision Control System) [8] or *bazaar* (Bazaar GPL Distributed Version Control Software) [12] work, as well as the proprietary system BitKeeper (Bitkeeper Source Management) [14], chosen by Linus Torvalds to maintain Linux since February 2002, since according to him there was no appropriate free tool. It is said that using Bitkeeper doubled the pace of development of Linux. Nonetheless, the decision came under heavy criticism because it was proprietary, with a licence that allowed free projects to obtain it free of charge on condition that all commit changes with their metadata were logged on a public server designated by the owners and accessible to everyone, and always on condition that the licensee did not try to develop another source control system to compete with it. It was precisely the attempt to develop a compatible free product by an employee of the same company where Linus Torvalds worked that detonated the change in source management system. Linus rapidly developed a provisional replacement, *git* ("Git manual page") [218], which soon became definitive, condensing all of the experience of Linux's cooperative and decentralised development: it supports large-size projects in a decentralised fashion, facilitating to a great extent the development of tentative branches and their merging with others or with the main one, with cryptographic security mechanisms that prevent altering the log. As of April 2005, Linux is maintained using *git* or its wraps (for example, *cogito* "Cogito manual page" [90].

8.6. Documentation

In the world of free software, WYSIWYG text processors and other office suite tools that are so successful in other environments are barely used, even though there are already free tools such as OpenOffice.org. This is due to several important factors:

- It is advisable to maintain documentation under source control, and source control systems, like CVS, although they admit binary formats, prefer transparent text formats that can be edited with a normal text editor and processed with tools developed for programs that allow us to see the differences between versions easily, to generate and apply patches based on those differences, and to carry out merges.

> **Note**
>
> In Unix the most common tools for these operations are *diff, diff3, patch* and *merge.*

- Some free documentation licences, especially the GFDL (*vid.* section 10.2.1), demand transparent formats, especially because they make the job easier for those who prepare derived documents.

- The WYSIWYG tools ("what you see is what you get") generally do not contain any information other than the strict visualisation, making it very difficult, if not impossible, to identify authors, or titles, or conversion to other formats. Even if they do allow conversion to other formats, this tends

to be done interactively, and is often impossible to automate (using *make*, for example).

• In general, office applications generate sizeable file formats, which is an undesirable feature for both developers and repositories.

For all of the above, free programmers, accustomed to programming and compiling, prefer transparent document formats, in many cases pure simple text and in many others processable document formats.

The processable formats in use are not many. Traditionally, in the world of Unix programs have been documented in the formats expected by the family of processors *roff*, with a free version (*GNU troff*) [37] by Norman Walsh. Nevertheless, this practice has been gradually abandoned, except for traditional manual pages, since it is almost obligatory to prepare manual pages for the system's most basic tools. Because many manual pages have grown so much so that it is barely appropriate to call them *pages*, it was necessary to prepare an alternative hypertext format that allowed documents structured with indexes and cross-references to be followed. The GNU project designed the *texinfo* format (Texinfo - The GNU Documentation System) [63] and made it its standard. This format allows navigable documents to be obtained with the *info* tool or within the *emacs* editor, and in turn, to obtain quality document printouts using the TeX text processor, of Donald Knuth (The TeXbook) [156].

The *texinfo* format can be translated into multipage HTML if required, and many people prefer to view the information with a web navigator, but the capacity to search for words in a document is lost. This is one of the unwanted results of the popularity of HTML, since the navigators do not implement the concept of *multipage document* , despite the fact that there are `link` elements that allow parts to be interlinked.

There is overwhelming demand for being able to view complex documents as easily navigable multipage web pages. There are people who write documentation in LaTeX (*LaTeX user's guide and reference manual*) [163], also a TeX application, very popular among scientists, more expressive than Texinfo and convertible to multipage HTML with certain tools (The LaTeX Web Companion) [130], on condition a certain discipline is maintained. Indeed, TeX applications are sets of macros that combine very low level typographic operators to convert them into abstract languages that work with high level concepts (author, title, summary, chapter, section, etc.). If we only use the basic macros, conversion is simple. But since nobody prevents the use of low level operators and, additionally, there are enormous quantities of macro packages beyond the maintenance capacity of conversion tool maintainers it is difficult to achieve good conversions.

8.6.1. DocBook

The problem stems from the fact that there is no distinction between content and presentation, either in TeX or in *nroff*, since the abstraction is built in layers. This distinction is made by SGML applications (*standard generalised markup language*) [81] and XML (*extensible markup language*) [224], where the presentation is specified with completely separate *style sheets*. Soon very simple SGML applications started to be used, such as `linuxdoc` and `debiandoc`, but due to their limited expressive capacity, `DocBook was chosen`. (*DocBook: the definitive guide*) [225].

`DocBook` is an SGML application originally developed for technical IT documentation and now has an XML variant. Currently, DocBook is the standard free documentation format for many projects (Linux Documentation Project, KDE, GNOME, Mandriva Linux, etc.) and a goal to be reached for others (Linux, *BSD, Debian, etc).

However, DocBook is a complicated language, plagued by tags, which means that it is useful to have tools to help with the editing, even if they are very basic and rare; one of the most popular tools of this type is the `psgml` mode of *emacs*. It is also heavy to process and free processors still generate a not very attractive quality of documents.

8.6.2. *Wikis*

Many people find it too complicated to write documentation with such complex languages as DocBook and collaboration mechanisms like CVS. This is why a new mechanism of collaboration for online document preparation via the web has become popular, called *wiki*, and invented by Ward Cunningham ("Wiki design principles") [97]. It was first put into service in 1995 and is now extensively used in a wide range of variants for preparing very dynamic documents, not designed for printing and often with a short life (for example, conference organisation).

Unlike DocBook, a *wiki* has a very simple and concise markup language which is reminiscent of the final presentation, without being exactly like it. For example, paragraphs are separated by a blank line, elements of a list are started with a hyphen if not numbered and with a zero if they are numbered, and table cells are separated by vertical and horizontal bars.

Neither does the concept of a "full document" exist, rather a *wiki* is more a set of small interlinked documents created as and when it is necessary to explain a new concept or subject. The documents are created almost automatically, as the editing tool shows very clearly that we have entered a concept (through a `WikiName`, almost always two joined words with the first letter capitalised). Hardly any *wiki* allow hyperlinks within the same page.

Unlike CVS, anyone can write in a *wiki*, although it is advisable for the author to identify himself by previously registering. When we visit a *wiki* we can see that all pages have a button that allows them to be edited. If pressed, the navigator will show us a form with the document's source code, which we will be able to change. This is not a WYSIWYG edit, which discourages anyone just wanting to interfere, but is simple enough for anybody interested to be able to modify documents with very little effort.

Wikis carry their own document version control, in such a way that all of their versions are generally accessible, indicating who made them and when. They can also be easily compared. Plus, they tend to include search mechanisms, at least per page name and word content.

Normally, the original author of a page will want to know what changes are made to it. To do so he can subscribe to the changes and receive notifications of them by e-mail. Sometimes, the person seeing a document will not dare to change anything, but may make a comment. Normally, all *wiki* pages have an associated comments forum pasted at the end of the document, which either the original author or anybody who assumes the role of editor can use to reform the original text, possibly by moving phrases from the comments to the relevant places.

Advice

The best way of understanding the *wiki* concept is to access one and experiment on a page designed for this purpose, usually called `SandBox`.

8.7. Bug management and other issues

One of the strong points of the free development model is that the community contributes with bug reports and feels that those reports or solutions are given attention. This requires a simple bug reporting mechanism, so that developers can receive sufficient information, in a systematic way and containing all necessary details, either provided by the collaborator, with an explanation of what is happening, the level of importance and possible solution, or through an automatic mechanism that determines, for example, the program version and environment in which it functions. Errors should also be saved in a database that can be consulted, to see whether a bug has already been communicated, corrected, its level of importance, etc.

There are several of these systems, with different philosophies. Some are via web, others via e-mail, through some intermediary program. They all have a web interface for consultation. Some allow anonymous reports, while others require identification (a valid e-mail address) to prevent noise. Although web procedures would appear to be the most simple, they do not easily obtain automatic information on the bug's environment. For example, the Debian system provides programs like reportbug, which after asking for the name of the package that we wish to report on, consults the error server for the bugs

reported to it. If none of them refers to our problem, we will be asked for a description of it, its level of importance ("critical", "grave", "serious", "important", "cannot be regenerated from source codes", "normal", "minor" or "suggestion") and labels about its category (for example, "security"). Following this, if we confirm the request, it will automatically find out the version of the package and those on which it depends, in addition to the kernel's version and architecture. Obviously, it knows the e-mail address, so it sends to the correct site a report similar to the following one:

```
Package: w3m-ssl Version: 0.2.1-4 Severity: important After reloading a page containing
complex tables several dozen times, w3m had used all physical memory and thrashingcommenced.
This is an Alpha machine. --System Information Debian Release: testing/unstable Kernel Version:
Linux romana 2.2.19 #1 Fri Jun 1 18:20:08 PDT 2001 alpha Unknown Versions of the packages
w3m-ssl depends on: ii libc6.1 2.2.3-7 GNU C Library: Shared libraries and Timezone data
ii libgc5 5.0.alpha4-8 Conservative garbage collector for C ii libgpmg1 1.19.3-6 General Purpose
Mouse Library [libc6] ii libncurses5 5.2.20010318-3 Shared libraries for terminal handling
ii libssl0.9.6 0.9.6a-3 SSL shared libraries ii w3m 0.2.1-2 WWW browsable pager with
tables/frames support
```

This message generates a bug number which is returned to us, sent to the maintainer and saved in the database. When the bug is solved, we will also receive a notification. Every bug has an e-mail address assigned to it that can be used to provide additional information, for example. We can consult the bug database http://bugs.debian.org at any time.

Sometimes bug monitoring systems have mechanisms for assigning someone to solve them and setting a deadline. There are also other issues, such as pending jobs, requested improvements, translations, etc., that require similar management mechanisms. With free software we cannot generally use very rigid mechanisms for managing the tasks that each developer has to do. After all, many collaborators are volunteers and cannot be obliged to do anything. Nonetheless, tasks can be defined and we can wait for somebody to subscribe to the system and to take them on within a declared period. Whether there is control over what certain people can do or not, it is always advisable to control all the tasks that need to be done, who and what they depend on, their level of importance, and who is working on them. Many important projects manage these aspects using Bugzilla (*The Bugzilla guide*) [89] or its derivatives.

Sometimes someone working on a project may discover a bug on a different project on which his work depends, but that has a different bug management system to the one to which he is accustomed. This is particularly true for users of distributions who wish to use a single tool for reporting and monitoring bug solving. To facilitate reporting and monitoring of those bugs, it may be advisable to *federate* different systems, as done by *Malone* (The Malone Bug Tracker) [47].

8.8. Support for other architectures

The minimum support required for working with a portable program is access to *compilation farms* , which allow the program to be compiled on different architectures and operating systems. For example, SourceForge (*vid.* section 8.9.1) offered for a time Debian GNU/Linux environments for Intel x86, DEC Alpha, PowerPC and SPARC, in addition to Solaris and Mac OS/X environments.

It is also useful to be able to test (not just compile) the program in those environments. But this service requires more resources and more of the administrator's time. The compilation service can already be problematic, because normally we need to provide compilation environments for several languages, with a large number of libraries. If what we want to do is to test any program, the difficulties increase exponentially, not just because it is very difficult to have the necessary resources available, but also for security reasons, which can make it extremely complicated to administrate those systems. Notwithstanding, there are a few *compilation farm* services, with standard installations of various architectures, which can allow us to test some things.

The abovementioned public farms are normally a service that requires manual use. The invited developer copies his files onto one of those machines, compiles them and tests the result. He will probably have to do it from time to time, prior to releasing an important version of the program. It could be much more interesting for compilations and the execution of regression tests to be carried out systematically, in an automated fashion, for example every night, if there have been changes in the source codes. This is how some important projects operate, which provide their own infrastructure for external developers, which tends to be called a *tinderbox*. This is the case with Mozilla, financed by Netscape, whose *tinderbox* (http://www.mozilla.org/tinderbox.html) [50] has a web interface to the results of the compilation and tests of the navigator's components on all of the architectures on which it operates. This interface is closely related to the CVS and shows those results for different states (between commits), identifying the one responsible for the bugs, and facilitating progress, by overcoming the problem until it is resolved. Tinderboxes are also used by the projects OpenOffice and FreeBSD, at least.

8.9. Development support sites

Development support sites offer, in a more or less integrated fashion, all of the services described above plus a few additional ones that allow projects to be searched by categories and to classify them according to some simple parameters of activity. This spares the developer having to set up and administer an entire infrastructure for collaboration, allowing him to concentrate on the project.

8.9.1. SourceForge

With regards to this type of service, one of the first to become established, and the most popular, is SourceForge (http://sourceforge.net) [61], managed by the OSDN (Open Software Development Network), a subsidiary of VA Software, which in March 2007 hosted more than 144,000 projects. It is structured around a set of programs with the same name, and which up to version 2 were free software.

SourceForge, as a prototype for this type of sites, offers a web interface or global access portal (http://sourceforge.net/) and a subportal per project (http://proyecto.sourceforge.net). The global interface shows news, advertisements, links, and an invitation to become a member or to enter if we already are members. To collaborate on the site, it is advisable to become a member, and it is compulsory if we want to create a new project or to participate in an existing one. To be a spectator it is not necessary, and as such, we can see what are the projects experiencing most active development or downloaded most frequently, and search for projects by category or descriptive word, and they will appear in order of activity level. For each project we can see its description, status (alpha, beta, production), its descriptors (programming language, operating system, subject, type of users, language, licence...), bugs and pending or reinstated aspects, activity levels over time..., or download it. We can also take part in forums or report bugs, even anonymously, which is not very advisable (because, for example, we may not get a reply).

Any authenticated user can request to register a project, which the administrators will admit on condition that it fulfils the site's policies, which in the case of SourceForge are fairly liberal. Once authorised, the creator can register other users as additional administrators or as developers, with access to modify the sources. Following authentication, there are not many more controls over the project, which means that there are a lot of dead projects. This does not confuse users too much though, because project searches sort the projects by level of activity, meaning that low or nil activity projects are barely visible. These projects run the risk of being eliminated by the site owners. The services that SourceForge offers a project, and that we could expect from any other similar service are as follows:

*   Hosting for the portal web pages of the project, at the address *project.sourceforge.net*, for viewing by the public. These pages can be static or dynamic (with CGI or PHP), in which case they can use a database (MySQL). They are entered directly through remote copy orders and can be handled using remote terminal interactive sessions (SSH).

*   Optionally, a virtual server that responds to addresses from a separately obtained domain, like *www.project.org* or *cvs.project.org*.

- As many web forums and/or mailing lists as necessary in the administrator's opinion.

- A news service where administrators announce advances concerning the project.

- *Trackers* for bug reporting and monitoring, requests for support, requests for improvements or integration of patches. Administrators give the issue a priority level and assign a developer to find the solution.

- Task managers, similar to trackers, that allow sub-projects to be defined with a series of tasks. These tasks, in addition to a priority level, are given a deadline. From time to time, developers assigned these tasks can show percentages of task completion.

- A CVS or Subversion with initial access rights for all developers.

- Uploading and downloading service for software packages. It registers entered versions when used and interested parties can receive a notification when this occurs. Plus, the initial upload involves the creation of several replicas worldwide, which facilitates distribution.

- Service for publishing documents in HTML format. Anyone can register them, but they will only be visible following approval by an administrator.

- Back-up copy for disaster recovery, such as broken drive, not user bugs, like accidentally deleting a file.

- Integrated mechanism for donations to users, to projects and to Source-Forge.

An authenticated user will have a personal page containing all relevant information, such as projects to which the user is associated, themes or tasks pending, as well as forums and files that he has said he wants to supervise. Plus, so that he does not have to be tending to his personal page, the user will receive notifications to his e-mail about the things he wishes to control.

8.9.2. SourceForge heirs

In 2001, VA Software was about to go bankrupt, in the full swing of the dot-com crisis. Then it announced a new version of its SourceForge software with a non-free licence, in an attempt to secure a source of revenue by selling it to companies for their internal developments. At the same time, it eliminated mechanisms that allowed a project to be dumped for moving to another site. Both events were seen as a threat that the thousands of projects hosted by SourceForge would become trapped in the hands of a single company, which would use the platform for showing non-free software. In the face of

this and the possibility of the site closing, offspring of the free version were developed and portals based on it were opened, particularly Savannah (http:// savannah.gnu.org) [57], dedicated to the GNU project and to other programs with *copyleft*-type licences, or BerliOS (BerliOS: The Open Source Mediator) [13], conceived as a meeting point for free software developers and companies. However, this is just a step in the direction of developing a distributed and replicated platform, where nobody has absolute control over the projects (Savannah The Next Generation, 2001) [98].

Another example of a free software project management system is Launchpad (https://launchpad.net) [43], used by Ubuntu for developing each version of the distribution. Launchpad is not a repository for source code, it is designed rather to offer support for monitoring code, incidents and translations. To achieve this it uses the already mentioned Malone tool, which allows incidents to be redirected to each code repository of the affected modules.

8.9.3.   Other sites and programs

Naturally, collaboration systems have been and continue to be developed, and some companies base their business on maintaining and servicing those sites. For example, the Tigris project (Tigris.org: Open Source Software Engineering Tools) [64], which not only maintains free software engineering projects, it also uses a collaboration portal (SourceCast) maintained by a service company (CollabNet), which also maintains individual projects' sites, like OpenOffice.org. Emerging new sites adopt new free software, such as GForce (http://gforge.org) [30], used by the Debian project (http://alioth.debian.org) [5]. We can see a detailed comparison of many sites in "Comparison of free/ open source hosting (FOSPhost) sites available for hosting projects externally from project owners" [202].

9. Case studies

> "GNU, which stands for 'Gnu's Not Unix', is the name for the complete Unix-compatible software system which I am writing so that I can give it away free to everyone who can use it. Several other volunteers are helping me. Contributions of time, money, programs and equipment are greatly needed."
>
> Richard Stallman, "The GNU Manifesto" (1985)

This chapter provides a more in-depth study of some of the most interesting free software projects in terms of the impact on the free software world, the results obtained, the management models, historical development, etc. Of course, the number of projects that we can discuss here is much smaller than the total number of free software projects (dozens of thousands), which means that this chapter should not be thought of as comprehensive, and neither can it ever be. Nevertheless, we hope that readers, having read the chapter, will at least have a basic understanding of how the theories that we have discussed throughout this book have been put into practice.

The projects that we have chosen range from lower-level applications, the ones which interact more with the computer's physical system rather than the user, to work environments designed for the end user. We have also included free software projects that, in principle, are not strictly development projects. This mainly applies to the distributions, which tend to be used as integrating systems, as they mainly take an extensive but limited set of independent applications and use them to create a system in which everything interacts effectively, including the options for installing, updating and deleting new applications, as desired by the user.

The lowest-level projects that we will look at will be Linux, the kernel of today's most popular free operating system and FreeBSD, which combines the kernel from the BSD family with a series of applications and utilities made by third parties. The work environments for end users that we will study will be KDE and GNOME, which are certainly the most widely-used and popular. For the servers, one of the main aspects in free systems, we will look at Apache, the leader in the WWW servers market, in this chapter. Likewise, we will introduce Mozilla, one of the WWW clients (it is in fact, much more than that) that we can rely on in the free software world. The last project that we will look at in this chapter is OpenOffice.org, a free Office IT (*suite*) package.

To finish off, we thought it would be appropriate to study the details of two of the most popular distributions, Red Hat Linux and Debian GNU/Linux, and to compare their sizes to other widely used systems, such as Microsoft Windows or Solaris.

After discussing the different case studies, we provide a table showing the most important characteristics of each application or project. One of the elements that readers will probably find most surprising will be the results of the cost and duration estimations and the number of developers required. We have obtained these results using methods typically used in the field of software engineering, especially the COCOMO Software Cost Estimation Model. The COCOMO model (*Software Engineering Economics*, 1981) [93] takes the number of source code lines as the starting measurement and generates estimates of the total cost, the development time and effort required to create the software. COCOMO is a model designed for "classical" software generation processes (waterfall or V model developments) and for average-size or large-scale projects; therefore, the figures that it will produce for some of the cases we analyse should be taken with some reservations. In any event, the results can help to give us an idea of the sheer scale on which we are working and of the amount of strenuous effort that would be necessary to achieve the same results with a proprietary software development model.

In general, it is the cost estimates that are most striking out of all the figures resulting from the COCOMO model. Two factors are taken into account in this estimate: a developer's average salary and the *overheads*. For calculating the estimated costs, the average salary for a full-time systems programmer is taken from the year 2000 "Salary survey 2000" [235]. The *overheads* are the extra costs that all companies must pay so that the product can be released, independently of the salary paid to the programmers. This ranges from the salaries of the secretaries and the marketing team to the costs of the photocopies, lighting, hardware equipment, etc. To summarise, the cost calculated by COCOMO is the total cost that a company would have to incur in order to create software of the specified dimensions and it should be remembered that only a part of this money would be received by the programmers for designing the software. Once this is factored in, the costs no longer seem so excessive.

9.1. Linux

The Linux kernel is, without a doubt, the star application of free software, to the extent that, whilst only constituting a small part of the system, its name is used to define the whole. Furthermore, it could even be said that free software itself is confused with Linux on many occasions, which is a pretty big mistake to make, given that there is free software that runs on systems not based on Linux (in fact, one of the biggest aims of the movement and of many free software projects is to create applications that can run in numerous environments). On another note, there are also applications that work in Linux and that are not actually free software (such as Acrobat Reader, the proprietary PDF documents reader, for which there is also a Linux version).

Note

There are actually various projects that integrate and distribute free applications that run on Windows systems, to avoid free software becoming associated solely with Linux systems. One of the pioneers in this area (and the one that probably became most well-known and comprehensive) was GNUWin, which was distributed on self-bootable CDs with more than a hundred free applications for Win32 systems. Most of these applications are also available in common GNU/Linux distributions, which made GNUWin a good tool for preparing for a gradual and easy transition from a Windows system to a GNU/Linux one. As at early 2007, there are other similar systems available, such as Win-Libre.

9.1.1. A history of Linux

The history of Linux is one of the most well-known histories within the world of free software, most probably because it has the traits of a legend rather than those of the history of a computer programme. In 1991, a Finish student called Linux Torvalds decided that he wanted to learn how to use protected mode 386 on a machine that his limited income had allowed him to purchase. At that time, there was a kernel in the operating system called Minix, designed for academic purposes and for use in university courses on operating systems; this is still used today. Andrew Tanenbaum, one of the most prestigious professors at the university, was the leader of the team working on the development of Minix, based on traditional Unix systems. Minix was a limited system, but quite capable and well-designed, and was at the centre of a large academic and engineering community.

Minix had a free distribution license and could be used for academic purposes, but it had the big disadvantage that people that did not work or study in the University of Amsterdam could not add improvements to it; instead these improvements had to be made independently, usually using patches. This meant that in practice, there was an official version of Minix that everybody used and then a long series of patches that had to be applied later to obtain additional functions.

In mid-1991, Linus, then an anonymous Finnish student, sent a message to the Minix newsgroup announcing that he was going to start work on an operating system kernel based on Minix, from scratch, rewriting code. At the time, although Linus did not explicitly say that he was going to publish it with a free software license, he noted that the system that he was going to create would not have the *barriers* that Minix had; this would indicate that, unbeknown to him, and probably without actually wanting to, he was taking the first step towards making the community that congregated around Minix at that time his.

Version 0.02, which dates from October 1991, despite being very limited, could already execute *bash* terminals and the GCC compiler. Over the course of the following months, the number of external contributions grew to the point that in March 1992, Linus could publish version 0.95, which was widely acknowledged as almost stable. There was still quite a way to go, however, before version 1.0, which is usually considered the first stable one. In December

1993, for example, version 0.99pl14 was published (which would make it the fourteenth corrected version of version 0.99); in March 1994, Linux 1.0 was finally born. By this time, Linux was being published under the terms of the GPL license; according to Torvalds himself, this was one of the best decisions he ever made, as it was extremely helpful in distributing and popularising his kernel. In "Evolution in open source software: a case study", [128] there is an exhaustive analysis of the evolution of the different versions of the Linux kernel, focusing on the scale and modularity.

Note

Another significant event in the annals of free software was the debate that took place in late January 1992 on the Minix newsgroup between Andrew Tanenbaum and Linus Torvalds. Tanenbaum, who was probably a bit annoyed by Torvalds' success with his "toy", attacked Linux and Linus in a rather disproportionate manner. His essential point was that Linux was a monolithic system (the kernel integrates all the handlers and the rest) and not a *microkernel* system (the kernel has a modular design, which means that it can be much smaller and that modules can be loaded upon demand). The original argument can be read just as it occurred in "The Tanenbaum-Torvalds debate" newsgroup [214].

9.1.2. Linux's way of working

The way Torvalds worked was not very common at that time. The development was mainly based on a mailing list[9]. The mailing list was a place where people not only argued, but where developments also took place. And this was because Torvalds was extremely keen on having the whole life of the project reflected on the mailing list, which is why he would ask people to send their patches to the list. Contrary to what one might have expected (the patches sent as attachments), Linus preferred to have the code sent in the body of the message so that he and others could comment on the code. In any case, although many people would provide their opinions and send corrections or new functions, the last word would always go to Linus Torvalds, who would decide on what code would be incorporated into Linux. To a large extent, this is still how it works in 2007.

[9]The list's email is linux-kernel@vger.kernel.org. The historical messages can be seen at http://www.uwsg.indiana.edu/hypermail/linux/kernel/.

Note

The consolidation of Linus Torvalds as a "benevolent dictator" has given rise to a large number of anecdotes within the project. For example, it is said that if an idea is liked, it must be implemented. If it is not liked, it must also be implemented. The corollary, therefore, is that good ideas are of no use whatsoever (without code, of course). On another note, if the implementation is not well-liked, it is essential to insist. A well-known case is that of Gooch, for whom Saint Job was a mere learner. Gooch made up to one hundred and forty six parallel patches until Linus finally decided to integrate them into the kernel's official branch.

Another one of Torvalds' innovative ideas was to develop two branches of the kernel in parallel: the stable one (the second number of the version is usually even, such as 2.4.18) and the unstable one (the second number of the version is odd, such as 2.5.12) As ever, Torvalds is the person that decides what goes into which branch (many of the most controversial decisions are related precisely to this point). In any case, Linux does not have any planned deliveries in fixed timeframes: it will be ready when it is ready and in the

meantime we'll just have to wait. Surely by now, most readers will have realised that the decision on whether the system is ready or not will be made solely by Linus.

The development method used in Linux has proven to be very effective in terms of results: Linux is very stable and any bugs are corrected extremely quickly (sometimes in minutes), as it has thousands of developers. In this situation, when there is a bug, the probability that someone will find it is very high, and if the person that discovers it is not able to correct it, someone will appear who will hit on the solution very quickly. To summarise, this shows how Linux has thousands of people working on its development every month, which is why its success is not altogether surprising.

It should be noted, however, that this way of working is very expensive where resources are concerned. It is not unusual for there to be many mutually-exclusive proposals for a new function or that a dozen patches are received for the same bug. In most cases, only one of the patches will finally be included in the kernel, which means that the rest of the time and effort put into the patches by the other developers will have all been in vain. Linux's development model is, therefore, a model that works very well in Linux but which not all projects can permit themselves.

9.1.3. Linux's current status

In early 2007, Linux was at version 2.6, which included, in terms of improvements made to version 2.4, NUMA (Non-Uniform Memory Access, used a lot in multiprocessors), new filesystems, improvements to communication in wireless networks and sound architectures (ALSA) and many other improvements (if you're interested in the details of the changes in respect of previous versions, you may consult "The wonderful world of Linux 2.6" [186]).

Linux's development model has undergone some changes over recent years. Although the development mailing list is still the *soul* of the project, the code no longer has to pass through the list, necessarily. One of the things that have contributed to this in a large way is BitKeeper, a proprietary system that performs revision control, developed by the company BitMover, strictly following Linus Torvalds' recommendations. The use of this proprietary tool generated a lot of controversy, in which Linus, true to form, demonstrated his pragmatism again, as for him and many others, the CVS version control system was obsolete. The disagreements were brought to an end with the development of *git*, a revision control system with similar features to BitKeeper that is currently used in Linux's development. More specifically, Linux's development process follows a pyramidal hierarchy, in which the developers propose patches, shared via mail between levels, which have to be accepted by the next level

up, formed by controller and file maintainers. The subsystem maintainers are on a higher level, whereas Linus Torvalds and Andrew Morton are on the top level and have the final say where the acceptance of patches is concerned.

To summarise, the following table provides an x-ray picture of the Linux project, showing how it now has more than five million lines of code and that it can therefore be included amongst the largest free software projects (along with Mozilla and OpenOffice.org). As to the estimates regarding the time it would take to design such a project and the average number of developers that would be necessary, we should note that the former is certainly much less than the time that Linux has been around. On the other hand, this is more than compensated by the latter detail, given that the average number of developers working full-time that would be necessary for such a project is higher than the number ever available to Linux.

Note

The cost estimate that COCOMO shows is in the range of 215 Million US Dollars, a sum that, if we put it in the context of everyday figures that we might think about, would be double what the best football clubs might pay for a great football star.

Table 4. Analysis of Linux

Website	http://www.kernel.org
Beginning of the project	First message on news.comp.os.minix: August 1991
License	GNU GPL
Analysed version	2.6.20 (stable version on 20/02/2007)
Source code lines.	5,195,239
Cost estimate (according to basic COCOMO)	$ 215,291,772
Design time estimates (according to basic COCOMO)	8.83 years (105.91 months)
Estimate of average number of developers (according to basic COCOMO)	180.57
Approximate number of developers	These are estimated in the thousands (although only hundreds appear in the credits [219])
Development assistance tools	Mailing list and *git*

Linux's composition in terms of programming languages shows a clear predomination of C, which is considered to be an ideal language for designing speed-critical systems. When speed is such a strict requirement that not even C can achieve it, assembly language is directly used for programming and this, as we can see, happens with some frequency. The disadvantage of this assembly language, in comparison with C, is that it is not as portable. Each architecture has its set of particular instructions, which means that a lot of code

written for an architecture in assembly language has to be ported to the other architectures. The incidence of the rest of the languages, as shown in the attached table, is marginal and they are limited to installations functions and development utilities. The version analysed for this book was Linux 2.6.20, as it was published on 20th February 2007 (without including any subsequent patches).

Table 5. Programming languages used in Linux

Programming language	Code lines	Percentage
C	4,972,172	95.71%
Assembler	210,693	4.06%
Perl	3,224	0.06%
Yacc	2,632	0.05%
Shell	2,203	0.04%

9.2. FreeBSD

As we have mentioned in the chapter on the history of free software, there are other types of free software operating systems, apart from the popular GNU/Linux. A family of these are the "inheritors" of the distributions of Berkeley University, in California (US): BSD type systems. The oldest and most well-known BSD system is FreeBSD, which was created in early 1993, when Bill Jolitz stopped publishing the unofficial updates to 386BSD. With the assistance of the company Walnut Creek CDROM, which subsequently changed its name to BSDi, a group of volunteers decided to carry on creating this free operating system.

The main objective of the FreeBSD project is the creation of an operating system that can be used without any type of obligations or ties, but that has all the advantages of code availability and is carefully processed to guarantee the quality of the product. The user has the liberty to do whatever they like with the software, either by modifying it according to their wishes or by redistributing it in an open form or even in a closed form, under the terms that they wish, with or without modifications. As the name itself indicates, the FreeBSD project is based, therefore, on the philosophy of BSD licenses.

9.2.1. History of FreeBSD

Version 1.0 appeared towards the end of 1993 and was based on 4.3BSD Net/2 and 386BSD. 4.3BSD Net/2 had code that was created in the seventies, when Unix was being developed by AT&T, which, as it turned out, involved a series of legal problems that were not resolved until 1995, when FreeBSD 2.0 was published without the original code developed by AT&T but based on 4.4BSD-

Lite, a *light* version of 4.4BSD (in which many of the modules had been eliminated for legal reasons, apart from the fact that the *port* for Intel systems was still incomplete) that was released by the University of California.

The history of FreeBSD would not be complete if we neglected to mention its "sister" distributions, NetBSD and OpenBSD. NetBSD appeared as version 0.8 in the middle of 1993. The main aim was for it to be very portable (although at the beginning it was only an adaptation for i386); consequently, the product's motto was: "Of course it runs NetBSD". OpenBSD arose from the division of NetBSD caused by philosophical differences (as well as personal differences) between developers in mid-1996. The focus is mainly on security and cryptography and they say that it is the safest operating system that exists, although, as it is based on NetBSD, it is also highly portable.

9.2.2. Development in FreeBSD

The development model used by the FreeBSD project is based mainly on two tools: The CVS version control system and the GNATS bug-tracking software. The whole project is based on these two tools, as is confirmed by the fact that a hierarchy has been created on the basis of these tools. In effect, it is the *committers* (the developers with write-access to the CVS repository) who have the most authority for the project, either directly or indirectly through the choice of the *core group*, as we shall see in the next section.

You do not have to be a *committer* in order to make bug reports in GNATS, which means that anyone who wishes to can report a bug. All the (*open*) contributions in GNATS are evaluated by a *committer*, who may assign the (*analysed*) task to another *committer* or request more information from the person that originally made the bug report (*feedback*). There are situations in which the bug has been solved for some recent branches, which will then be specified with the *suspended* status. In any case, the goal is that the report should be *closed*, once the error has been fixed.

FreeBSD distributes its software in two forms: on the one hand, the ports, a system that downloads the source codes, compiles them and installs the application in the local computer, and on the other, the packages, which are simply the source codes of the precompiled ports and, therefore, in binary. The most important advantage of the ports over the packages is that the former allow the user to configure and optimise the software for their computer. On the other hand, in the package system, as they are already precompiled, it takes much less time to install the software.

9.2.3. Decision-making process in FreeBSD

The board of directors of FreeBSD, famously called the *core team*, is in charge of defining the direction of the project and ensuring that the objectives are met, as well as mediating in cases in which there are conflicts between *committers*.

Until October 2000, it was a closed group, which could only be joined by an express invitation from the *core team* itself. As of October 2000, the members are elected periodically and democratically by the *committers*. The most important rule for the election of the *core team* is as follows:

1) The *committers* that have made at least one *commit* over the last year have the right to vote.

2) The Board of Directors will be renewed every two years.

3) The members of the board of directors may be "expelled" by a vote of two thirds of the *committers*.

4) If the number of members of the board of directors is less than seven, new elections will be held.

5) New elections are held when a third of the *committers* vote for this.

6) Any changes in the rules require a quorum of two thirds of the *committers*.

9.2.4. Companies working around FreeBSD

There are numerous companies that offer services and products based on FreeBSD, which FreeBSD lists on the project's website. In this presentation of FreeBSD we will learn more about the most significant aspects: BSDi and Walnut Creek CDROM.

FreeBSD was born partly due to the foundation of the company BSDi in 1991 by the people from CSRG (Computer Systems Research Group) of the University of Berkeley, which would provide commercial support for the new operating system. Apart from the commercial version of the FreeBSD operating system, BSDi also developed other products, such as an Internet server and a gateway server.

Walnut Creek CDROM was created with the aim of commercialising FreeBSD as the final product, in such a way that it could be considered as a distribution in the style of those that exist for GNU/Linux, but with FreeBSD. In November 1998, Walnut Creek broadened its horizons with the creation of the FreeBSD Mall portal, which would commercialise all types of products based on FreeBSD (from the distribution itself to t-shirts, magazines, books, etc.), and announce third-party products on its website and provide professional FreeBSD support.

In March 2000, BSDi and Walnut Creek merged under the name BSDi to work together against the Linux phenomenon, which was clearly leaving BSD systems in general and FreeBSD particularly, standing in the shadows. A year lat-

er, in May 2001, Wind River purchased the part that was dedicated to generating the BSDi software, with the clear intention of boosting the development of FreeBSD for its use in embedded systems and intelligent devices connected to the Network.

9.2.5. Current status of FreeBSD

According to the latest data from the poll that Netcraft performs periodically, the number of web servers that run FreeBSD is approximately two million. A new user who wished to install FreeBSD could choose between version 6.2 (which could be considered as the "stable" version) or the more advanced or "development" branch. Whilst the former provides more stability, especially in areas such as symmetric multiprocessing, which has been completely redeveloped in the newer versions, the latter allows users to enjoy the latest developments. It is also important to bear in mind that the developed versions tend to include test code, which slightly affects the system's speed.

One of the star features of FreeBSD is what is known as the *jails*. The jails minimise the damage that might be caused by an attack on basic network services, such as Sendmail for the emails or BIND (Berkeley Internet Name Domain) for name management. The services are placed in a jail so that they run in an isolated environment. The jails can be managed using a series of utilities included in FreeBSD.

9.2.6. X-ray picture of FreeBSD

As we have mentioned in this last section, FreeBSD's functions are not restricted solely to developing an operating system kernel, but also include the integration of a multitude of utilities that are distributed together in the style of the GNU/Linux distributions. The fact that the development process of FreeBSD is very closely linked to the CVS versions control system means that by studying the system, we can obtain a good idea of what FreeBSD consists of. The figures shown below correspond to the FreeBSD analysis performed on 21st August 2003.

One of the most interesting aspects of FreeBSD is that the figures are very similar to the ones of in KDE and GNOME: the size of the software easily exceeds five million lines of code, the number of files is approximately 250,000 and the total number of *commits* is approximately two million. However, it is interesting to observe that the main difference between GNOME and KDE with respect to FreeBSD is the *age* of the project. FreeBSD recently made to its tenth year and it has been around for almost twice as long as the desktop environments with which we are comparing it. That the size should be similar, despite the fact that the development period must have been longer, is partly due to the fact that FreeBSD did not attract as many developers. There is a list of approximately four hundred developers with write-access to the CVS (*committers*), whereas the number of developers listed in the FreeBSD manual is

approximately one thousand. This is why the activity registered in FreeBSD's CVS is lower than the average (five hundred *commits* per day) than that registered in both GNOME (nine hundred) and KDE (one thousand seven hundred, including the automatic *commits*).

We have considered as the basic system of FreeBSD all that is placed in the *src/src* directory of the *root* module of the CVS. The activity that has been registered in the basic system over the last ten years is that of more than half a million *commits*. There are more than five million lines of code, although we should remember that this does not only include the kernel, but many additional utilities, including games. If we take only the kernel into account (which is in the subdirectory *sys*), the scale is of 1.5 million of source code lines, predominantly in C.

It is interesting to consider how the time estimate given by COCOMO corresponds exactly to the FreeBSD project's real time, although the estimate of the average number of developers is much higher than the effective number. We should also point out that in the last year, only seventy five *committers* have been active, whereas COCOMO supposes that over the ten years of development, the number of developers should be 235.

Finally, we must remember, as we have mentioned, that FreeBSD's main activity is based around the CVS repository and the bug control system GNATS activities.

Table 6. Analysis of FreeBSD

Website	http://www.FreeBSD.org
Beginning of the project	1993
License	Of BSD type
Analysed version	4,8
Source code lines.	7,750,000
Lines of source code (kernel only)	1,500,000
Number of files	250,000
Cost estimate	$ 325,000,000
Runtime estimate	10.5 years (126 months)
Estimate of average number of developers	235
Approximate number of developers	400 *committers* (1,000 collaborators)
Number of *committers* active in the last year	75 (less than 20% of the total)
Number of *committers* active in the last two years	165 (approximately 40% of the total)
Number of *commits* in the CVS	2,000,000

Average number of *commits* (total) per day	Approximately 500
Development assistance tools	CVS, GNATS, mailing list and news site

C is the predominant language in FreeBSD and it keeps a greater distance from C++ than the other case that we have studied in this chapter. It is interesting to note that the number of lines of code in the assembly language contained in FreeBSD, matches, in terms of the scale, those of Linux, although those corresponding to the kernel are only twenty five thousand, in total. To summarise, we could say that in FreeBSD, the more *classical* languages within free software, C, Shell and Perl predominate, and that the other languages that we have looked at in other applications and projects, such as C++, Java, Python, have not been integrated.

Table 7. Programming languages used in FreeBSD

Programming language	Code lines	Percentage
C	7,080,000	92.0%
Shell	205,000	2.7%
C++	131,500	1.7%
Assembler	116,000	1.5%
Perl	90,900	1.20%
Yacc	5,800	0.75%

9.2.7.   Academic studies on FreeBDS

Despite certainly being a very interesting project (we can look at its history by analysing the versions system, going back up to 10 years!), FreeBSD has not inspired that much interest in the scientific community. There is, however, one research team that has shown interest in the FreeBSD project, from various points of view ("Incremental and decentralised integration in FreeBSD") [149], which has especially focused on how software integration problems are resolved in an incremental and decentralised fashion.

9.3.   KDE

Although it was not the first solution in terms of user-*friendly* desktop environments, the dissemination of the Windows 95 operating system in mid-1995 caused a radical change in the way non-specialised users interacted with computers. From the one-dimensional systems of lines of instructions (the terminals), the metaphor of the two-dimensional desktop environment was born, where the mouse began to be used more than the keyboard. Windows 95, more than a technological innovation, must be credited as being the system

that managed to cover all the personal and office environments, setting the standards that would be followed in the future (technical and social rules that, we are still, in some cases, suffering from in the early 21st Century).

Before desktop systems were created, each application managed its own appearance and manner of interacting with the user, autonomously. On desktops, however, the applications must have common properties and an appearance that is shared by the applications, which eases the pressure on the user, who can *reuse the way of interacting* learnt whilst using one application, when using another. This also eased the pressure on the application developers, as they did not have to deal with the problem of creating the interactive elements starting from zero (which is always a complicated task), but could start from a predefined framework and predefined rules.

9.3.1. History of KDE

Unix followers were quick to notice the outstanding success of Windows 95 and, given that Unix-like environments did not have systems that were as intuitive whilst still being free, they decided to get to work. The KDE K Desktop Environment project was born from this effort in 1996; it was designed by Matthias Ettrich (creator of LyX, an editing program in the TeX typeset) and other *hackers*. The KDE Project proposed the following aims:

> **Note**
>
> Originally, the name KDE stood for Kool Desktop Environment, but it was subsequently changed simply to K Desktop Environment. The official explanation was that the letter K is just before the L, for Linux, in the Roman alphabet.

* To provide Unix-like systems with a user-friendly environment that was, at the same time, open, stable, trustworthy and powerful.

* To develop a set of libraries for writing standard applications on a graphical system for Unix X11.

* To create a series of applications that would allow the user to achieve their objectives effectively and efficiently.

A controversy was created when the members of the newly-created KDE project decided to use a library orientated to objects called Qt, belonging to the Norwegian firm Trolltech, which was not covered under any free software license. It turned out that, although the KDE applications were licensed under GPL, they linked with this library, which meant that it was impossible to redistribute them. Consequently, one of the four freedoms established by Richard Stallman in the Free Software Manifesto was being violated [117]. As of version 2.0, Trolltech distributes Qt under a dual license that specifies that if the application that uses the library operates under the GPL, then the license valid for Qt is the GPL. Thanks to this, one of the most heated and hot-tempered debates in the world of free software had a happy ending.

9.3.2. Development of KDE

KDE is one of the few free software projects that generally follows a new version launch schedule (let us remember, for example, that there will be a new Linux version "when it is ready", whereas, as we shall discuss later, GNOME has always suffered significant delays when it came to releasing new versions). The numbering of the new versions follows a perfectly defined policy. The KDE versions have three version numbers: one higher one and two lower ones. For example, in KDE 3.1.2, the higher number is the 3, whereas the 1 and 2 are the lower numbers. Versions with the same higher number have binary compatibility, which means that it is not necessary to recompile the applications. Until now, the changes in the higher number occurred in parallel with the changes in the Qt library, which shows how the developers wanted to take advantage of the new functionalities in the Qt library in the imminent version of KDE.

Where the lower numbers are concerned, the versions with one single lower number are versions in which they have included both the new functionalities and in which the bugs that have been discovered, have been corrected. The versions with a second lower number do not include new functionalities in respect of the versions with the first lower number, and only contain the bug corrections. The following example will explain this better: KDE 3.1 is a third-generation version of KDE (higher number 3) to which new functionalities have been added, whereas KDE 3.1.1 is the previous version with the same functionalities, but with all the bugs that have been found corrected.

KDE was built, shortly after the project began, in an association registered in Germany (KDE e.V.) and, as such, the articles of association meant that there has to be a managing committee. The influence of this managing committee on the development is nil, as its main task is the administration of the association, especially where the donations that the project receives are concerned. In order to promote and disseminate KDE, the KDE League, which includes all interested companies, was created, as we shall discuss below.

9.3.3. The KDE League

The KDE League is a group of companies and individuals from KDE that have the objective of enabling the promotion, distribution and development of KDE. The companies and individuals that participate in the KDE League do not have to be directly involved in the development of KDE (although the members are encouraged to get involved), but simply represent an industrial and social framework that is friendly to KDE. The aims of the KDE League are as follows:

• Promoting, providing and facilitating the formal and informal education of the functionalities, capabilities and other qualities of KDE.

- To encourage corporations, governments, companies and individuals to use KDE.

- To encourage corporations, governments, companies and individuals to participate in the development of KDE.

- To provide knowledge, information, management and positioning around KDE in terms of its use and development.

- To foster communication and cooperation between KDE developers.

- To foster communication and cooperation between KDE developers and the general public through publications, articles, websites, meetings, participation in conferences and exhibitions, press articles, interviews, promotional materials and committees.

The companies that participate in the KDE League are mainly distribution designers (SuSE, now part of Novell, Mandriva, TurboLinux, Lindows and Hancom, a Korean free software distribution), development companies (Trolltech and Klarälvdalens Datakonsult AB), plus the giant IBM and a company created with the aim of promoting KDE (KDE.com). Of all these, we must especially mention Trolltech, Novell and Mandriva Software, whose involvement has been essential and whose business models are very closely linked to the KDE project:

- Trolltech is a Norwegian company based in Oslo that develops Qt, the library that can be used as a graphic interface for the user and an API for developers, although it can also work as an element embedded in PDA (such as the Sharp Zaurus). The importance of the KDE project for Trolltech is evidenced by two basic elements in its commercial strategy: on the one hand, it recognises KDE as its main promotion method, encouraging the development of the desktop and accepting and implementing the proposed improvements or modifications; on the other hand, some of the most important KDE developers work professionally for Trolltech; the most well-known example is that of Matthias Ettrich himself, who founded the project, which doubtlessly benefits both the KDE project and the company itself. Trolltech's involvement in the KDE project is not exclusively limited to the Qt library, as is proven by the fact that one of the main developers of KOffice, KDE's office software package, currently has a part-time contract with them.

- SuSE (now part of Novell) has always shown its special predilection for the KDE desktop system, partly due to the fact that most of its developers are of German or Central European origin, as is the company itself. SuSE, aware of the fact that the better and easier the desktop environment that its distribution offers, the greater its implementation and, therefore, the sales and support requests, has always had a very active policy in terms

of the budget allocated to professionalising key positions within the KDE project. As an example, the current administrator of the version control system and another two of the main developers are all on SuSE's payroll. Likewise, within SuSE's workforce there are a dozen developers that can spend some of their working time on developing KDE.

- The Mandriva distribution is another one of the biggest backers of KDE and a number of the main developers work for it. Its financial situation, which has included bankruptcy from 2003, has meant that it has lost influence over the last few years.

9.3.4.   Current status of KDE

After the publication of KDE 3 in May 2002, the general opinion is that the free desktops are on a par with their proprietary competitors. Some of its greatest achievements include the incorporation of a components system (KParts) that makes it possible to embed some applications in others (a piece of a KSpread spreadsheet in the KWord word processor) and the development of DCOP, a simple system for processes to communicate with each other, with authentication. DCOP was the project's commitment that acted in detriment to the CORBA technologies, a widely-debated subject within the world of free desktops, especially for GNOME, where it was decided that CORBA and KDE technologies would be used. History seems to have put each technology in its place, as can be seen from the DBUS proposal (an improved type of DCOP) from FreeDesktop.org, a project interested in promoting the interoperability and the use of joined technologies in free desktops, which is, coincidentally, led by one of the most well-known GNOME *hackers*.

The following table summarises the most important characteristics of the KDE project. The licenses that the project accepts depend on whether they are for an application or a library. The library licenses provide greater "flexibility" for third parties; in other words, they make it possible for third parties to create proprietary applications that are linked to the libraries.

The latest KDE version is, as at early 2007, version 3.5.6 and the fourth generation, KDE 4, which will be based on Qt4, is expected to arrive in mid-2007. The generation change involves a lot of effort on adapting the version, which is a tedious and time-costly task. However, this does not mean that the "old" applications will no longer work. Generally, expecting they will continue to work, the older versions of the libraries on which they were based are also included, although this means that various versions of the libraries have to be loaded simultaneously in the memory, with the ensuing waste of system resources. The KDE developers view this effect as an inherent part of the development of the project and, therefore, as a lesser evil.

9.3.5. X-ray picture of KDE

Where the scale of KDE is concerned, the figures that we will now discuss correspond to the status of CVS in August 2003, which means that they should be taken with the usual reservations that we have already discussed, plus one more: some of the modules that have been used in this study are still under development and do not fulfil the criteria of being a finished product. This shouldn't really have any effect for our purposes, as we are more interested in the scale of the results than the exact numbers.

The source code included in KDE's CVS is in the total sum of six million lines of code in different programming languages, as we shall show below. The time required to create KDE would be approximately nine and a half years, which is more than the project's seven years, and the estimated average number of developers working full-time would be two hundred. If we take into account the fact that KDE had approximately eight hundred people with write-access to CVS in 2003 (of which half have been inactive over the last two years) and the fact that the number of KDE developers with full-time contracts has not been more than twenty at any given time, we can see that KDE's level of productivity is much, much higher than the estimate provided by COCOMO.

Note

A company that wanted to develop a product of this scale starting from zero would have to invest more than 250 million dollars; for comparative purposes, this sum would be equivalent to an investment of a car manufacturer in the creation of a new production plant in Eastern Europe or what a well-known oil company is planning to spend in order to open two hundred petrol stations in Spain.

It is also interesting to see that a large part of the effort, almost half of that expended on the development of the KDE project, would correspond to the translation of the user interface and the documentation. Although very few (approx. one thousand) of the programming lines are concerned with this task, the number of files dedicated to this purpose is seventy five thousand for translations (a sum that increases to one hundred thousand if we include the documentation in the different formats), which comprises almost a fourth (third) of the 310,000 files that there are in CVS. The combined activity of CVS is of one thousand two hundred *commits* per day, which means that the average time between *commits* is approximately one minute[10].

[10]Two observations should be made on this point: the first is that when a *commit* including various files is made, it is as if there had been one separate *commit* for each file; the second is that the number of *commits* is an estimated sum, as the project has a series of *scripts* that perform *commits* automatically.

Where the tools, the information locations and the development assistance events are concerned, we will see that the range of possibilities offered by KDE is much wider than that used in Linux. Apart from the version control system and the mailing lists, KDE has a series of websites providing information and technical and non-technical documentation on the project. There is also a news site among these sites where new solutions are presented and proposals are debated. The news site, however, cannot be considered as a replacement for the mailing lists, which, as occurs with Linux, is where the real debates take place and the decisions are made and the future strategies devised; the

news site is really more of a meeting point for the users. Finally, KDE has been organising regular meetings for three years, in which the developers and the collaborators meet for approximately a week to present the latest innovations, develop, debate and get to know each other and have a good time (not necessarily in that order).

Table 8. KDE Analysis

Website	http://www.kde.org
Beginning of the project	1996
License (for applications)	GPL, QPL, MIT, Artistic
License (for libraries)	LGPL, BSD, X11
Analysed version	3.1.3
Source code lines.	6,100,000
Number of files (code, documentation, etc.)	310,000 files
Cost estimate	$ 255,000,000
Runtime estimate	9.41 years (112.98 months)
Estimate of average number of developers	200.64
Approximate number of developers	Approximately 900 *committers*
Number of *committers* active in the last year	Around 450 (approximately 50% of the total)
Number of *committers* active in the last two years	Around 600 (approximately 65% of the total)
Approximate number of translators (active)	Approximately 300 translators for more than 50 languages (including Esperanto).
Number of *commits* (by developers) in the CVS	Approximately 2,000,000 (estimated figure not including automatic *commits*)
Number of *commits* (by translators) in the CVS	Approximately 1,000,000 (estimated figure not including automatic *commits*)
Average number of *commits* (total) per day	1,700
Tools, documentation and development assistance events	CVS, mailing lists, website, news site, annual meetings

Where the programming languages used in KDE are concerned, C++ predominates. This is mainly due to the fact that this is the native language of Qt, although a great effort is expended on providing bindings to allow developments in other programming languages. Certainly, the number of lines of code in the minority languages corresponds almost integrally to those bindings, although this does not mean that they are not used at all, as there are numerous projects external to KDE that use them.

Table 9. Programming languages used in KDE

Programming language	Code lines	Percentage
C++	5,011,288	82.05%
C	575,237	9.42%
Objective C	144,415	2.36%
Shell	103,132	1.69%
Java	87,974	1.44%
Perl	85,869	1.41%

9.4. GNOME

The main objective of the GNOME project is to create a desktop system for the end user that is complete, free and easy to use. Likewise, the idea is for GNOME to be a very powerful platform for developers. The initials GNOME stand for *GNU Network Object Model Environment*. From its name, we see that GNOME is part of the GNU project. Currently, all the code contained in GNOME must be under a GNU GPL or a GNU LGPL license. We can also see that the networks and the object-orientated modelling are extremely important.

9.4.1. History of GNOME

Whilst the freedom of KDE was still being argued about, in the summer of 1997, as fate would have it, Miguel de Icaza and Nat Friedman met at Redmond during some workshops organised by Microsoft. It is probable that this meeting caused a radical turnaround in both, resulting in the creation of GNOME by Miguel de Icaza when he returned to Mexico (along with Federico Mena Quintero) and his admiration for distributed object technologies. De Icaza and Mena decided to create an environment that would be an alternative to KDE, as they understood that a reimplementation of a proprietary library would have been a task destined to failure. And thus GNOME was born.

Since those ancient times in 1997, GNOME has gradually grown and continues to grow, with its repeated publications. Version 0.99 was launched in November 1998, but the first really popular version, distributed practically with any GNU/Linux distribution, would be GNOME 1.0, published in March 1999. It should be noted that the experience of this first *stable* version of GNOME was not very satisfactory, as many considered it to be full of critical bugs. For this reason, GNOME October (GNOME 1.0.55) is treated as the first version of the GNOME desktop environment that was truly stable. As we can observe, with GNOME October, the developers did not use numerated publication version so as to avoid entering a "versions race" against KDE. The first GUADEC, the GNOME users and developers European conference, was held in Paris in 2000 and narrowly missed coinciding with the publication of the

new version of GNOME, called GNOME April. It was the last version to be named after a month, as it turned out that this system created more confusion than anything else (for example, GNOME April was launched after GNOME October, although one could be forgiven for assuming the opposite). In October of that year, after numerous debates over the months in different mailing lists, the GNOME Foundation, which we shall discuss in subsequent sections, was created.

GNOME 1.2 was a step forward in terms of the architecture used by GNOME, an architecture that continued to be used in GNOME 1.4. This era was characterised by the second GUADEC, which took place in Copenhagen. What had begun as a small meeting of a few *hackers*, became a great event that captured the attention of the whole software industry.

In the meantime, the argument about the freedom of KDE was resolved with Trolltech's change of position, when it ended up licensing Qt under a dual license, which was for free software for applications that work with free software. Today, there is no doubt that both GNOME and KDE are free desktop environments, which means that we can say that the development of GNOME has encouraged the creation of not just one free desktop environment, but two.

9.4.2. The GNOME Foundation

The most difficult problem to take on board when you hear about GNOME for the first time is the organisation of the more than one thousand contributors to the project. It is paradoxical that a project with a structure that tends toward the anarchic, should be this successful and achieve complex objectives that only a few multinationals in the IT sector would be able to achieve.

Although GNOME was created with the clear aim of providing a user-friendly and powerful environment, to which new programs would gradually be added, it soon became apparent that it would be necessary to create a body that would have certain responsibilities that would allow them to promote and boost the use, development and dissemination of GNOME: consequently, the GNOME Foundation was created in 2000; its headquarters are situated in Boston, US.

The GNOME Foundation is a non-profit organisation and not an industrial consortium; it has the following functions:

• Coordinating the publications.

• Deciding which projects are part of GNOME.

- It is the official spokesperson (for the press and for both commercial and non-commercial organisations) of the GNOME project.

- Promoting conferences related to GNOME (such as the GUADEC).

- Representing GNOME in other conferences.

- Creating technical standards.

- Promoting the use and development of GNOME.

In addition, the GNOME Foundation receives financial funds for promoting and boosting the functions mentioned above, as this was impossible to do in a transparent manner before the foundation was created.

Currently, the GNOME Foundation has one full-time employee that is in charge of solving all the bureaucratic and organisational tasks that have to be done in a non-profit organisation that holds regular meetings and conferences.

In general terms, the GNOME Foundation is divided into two large committees: a managing committee and an advising committee.

The managing committee (the *Board of Directors*) is formed, at the most, by fourteen members elected democratically by the members of the GNOME Foundation. A "meritocratic" model is followed, which means that, in order to be a member of the GNOME Foundation, one has to have cooperated in one way or another with the GNOME project. The contribution does not necessarily have to involve source code; there are also tasks that require translation, organisation, dissemination, etc, which one could perform and then apply for membership of the GNOME Foundation, in order to have the right to vote. Therefore, it is the members of the Foundation that can put themselves forward for the board of directors and it is the members that, democratically, choose their representatives on the board from the persons that have put themselves forward. Currently, voting is by email. The duration of the term as member of the board of directors is one year, after which elections are held again.

There are some basic regulations for guaranteeing the transparency of the board of directors. The most remarkable one is the limitation on the number of members affiliated to the same company, which cannot exceed four employees. It is important to emphasise that the members of the board of directors are always so in their personal capacity, and never in representation of a company. Nevertheless, after a long discussion, it was agreed that this clause would be included to avoid any mistrust.

The other committee within the GNOME Foundation is the advising committee, which has no authority to make decisions but that serves as a vehicle for communicating with the managing committee. It is formed by commercial companies working in the software industry, as well as non-commercial organisations. Currently, its members include Red Hat, Novell, Hewlett-Packard, Mandrake, SUN Microsystems, Red Flag Linux, Wipro, Debian and the Free Software Foundation. All companies with more than ten employees are required to pay a fee in order to be part of the board of advisors.

9.4.3. The industry working around GNOME

GNOME has managed to enter the industry substantially, in such a way that various companies have participated very actively in its development. Of all of these, the most important cases are those of Ximian Inc., Eazel, the RHAD Labs by Red Hat and, more recently, SUN Microsystems. We will now describe, for each case, the motivations of the companies as well as their most important contributions to the GNOME desktop environment:

- Ximian Inc. (originally called Helix Inc.) is the name of the company that was founded in 1999 by Miguel de Icaza, the cofounder of GNOME, and Nat Friedman, one of GNOME's *hackers*. The main objective was to bring together the most important GNOME developers under the same umbrella to maximise development, which is why it is not surprising that its current and past employees have included around twenty of the most active GNOME developers. The application that Ximian put the most effort into from the very start was Evolution, a complete personal information management system in the style of Microsoft Outlook, which included an email client, agenda and a contacts address book. The products that Ximian sold were the Ximian Desktop (a version of GNOME with more corporate purposes), Red Carpet (mainly, although not limited to, a GNOME software distribution system) and finally MONO (a reimplementation of the .NET development platform), although the latter project is not, as yet, related in any way to GNOME. Ximian also developed an application that permits Evolution to interact with an Exchange 2000 server. This application, whilst being quite small, was very controversial because it was published with a non-free license (subsequently, in 2004, this component was also licensed as free software). In August 2003, Novell, as part of its strategy for entering the GNU/Linux desktop, bought Ximian.

- Eazel was founded in 1999 by a group of people who used to work for Apple, with the aim of making the GNU/Linux environment as easy as the Macintosh environment. The application on which they concentrated their efforts was called Nautilus and it was supposed to be the file manager that would definitively retire the mythical Midnight Commander, developed by Miguel de Icaza. The lack of a business model and the dotcoms crisis, which caused the risk investors to remove all the capital that was required for the company to carry on working, resulted in Eazel declaring

bankruptcy on 15th May 2001 and closing its doors. It did have time to release Nautilus version 1.0 before this however, although the numbering was rather artificial, given that the stability that one would expect in a 1.0 version was nowhere to be seen. Two years after Eazal's bankruptcy, we were able to see how Nautilus had developed and become a complete and manageable file manager integrated in GNOME; this means that the story of Eazel and Nautilus can be considered as a paradigmatic case of a program that survives the disappearance of the company that created it; something that is almost only possible in the world of free software.

- Red Hat created the Red Hat Advanced Development Labs, RHAD, with the aim of ensuring that the GNOME desktop would gain user-friendliness and power. In order to achieve this, Red Hat used half a dozen of the most important *hackers* from GNOME and gave them the freedom to develop whatever they decided was appropriate. From the RHAD Labs we have ORBit, the implementation of CORBA used by the GNOME project, known as "the fastest in the west". Another important aspect is the work that was carried out on the new version of GTK+ and on GNOME's configuration system, GConf.

- SUN Microsystems became involved in the development of GNOME at a later stage, as GNOME was a relatively mature product by September 2000. SUN's intention was to use GNOME as the desktop system for the Solaris operating system. It therefore created a team to work with GNOME, whose most important merits include the usability and accessibility of GNOME. In June 2003, SUN announced that it would distribute GNOME 2.2 with version 9 of Solaris.

9.4.4. GNOME's current status

GNOME, as at early 2007, is at version 2.18. Most of the key technologies on which it is based have matured, as is evident from the version number. For example, the CORBA *broker* used now is ORBit2, whilst the graphical environment and API, GTK+, underwent changes devised from the experience accumulated during the previous versions of GNOME. One important novelty is the inclusion of an accessibility library, proposed by SUN, which allows people with accessibility problems to use the GNOME environment. A special mention should also go to Bonobo, the GNOME components system. Bonobo left its mark on an era within GNOME, whilst the personal information management program Evolution was being developed. However, time has proven that the expectations raised by Bonobo were too high and that the reuse of the effort expended on it by employing its components has not been as extensive as was initially expected.

Note

The ATK library is a library of abstract classes that makes the applications accessible. This means that people with some form of disability (the blind, the colour-blind, people with eye problems, those who cannot use a mouse, a keyboard, etc.) may still use GNOME. SUN's interest on ensuring accessibility is due to the fact that if it wishes to offer its services to the government of the United States, it has to meet a series of accessibility standards. They have taken this work so seriously that there is even a blind programmer in the GNOME development team working at SUN. In September 2002, GNOME's accessibility architecture was given the Helen Keller Achievement Award.

9.4.5. X-ray picture of GNOME

The data and figures shown in table 10 bring us to the end of our presentation of GNOME. The figures correspond to the status of GNOME's CVS as at 14th August 2003. On that date, there were more than nine million lines of code hosted in the CVS repository owned by the GNOME project. Even though the most natural thing would be to compare GNOME to KDE, we must warn readers that the differences in terms of how these projects are organised make this unadvisable if we wish to make the comparison in equal conditions. This is due, for example, to the fact that GNOME's CVS includes GIMP (a program for creating and handling graphics), which, on its own, represents more than 660,000 lines of code, or the GTK+ library, on which GNOME's development focuses, and which, on its own, has 330,000 lines. If we add to this the fact that GNOME's CVS repository is more inclined to open new modules for programs (it has a total of seven hundred) than KDE's (which has less than one hundred), we can understand why GNOME has more lines than KDE, despite being a year and a half younger. The GNOME repository hosts more than 225,000 files, which have been added and modified almost two million times (*see* the number of *commits* some rows below, in the table).

Note

A company that wanted to create software of the size of GNOME's software, would have to contract an average of approximately two hundred and fifty developers for more than eleven years, in order to obtain a product with a similar extension, according to the CO-COMO model used throughout this chapter. The associated cost would be approximately 400 million dollars, a figure similar to that which a well-established mobile telephone company invested in 2003 to reinforce its network capacity, or similar to the figure that a car manufacturing firm would pay in order to open a production plant in Barcelona.

GNOME's human resources include almost one thousand developers with write-access to the CVS revision control system, of which almost twenty work for GNOME professionally (full-time or part-time). Of these, only 25% have been active in the last year and 40% have been active over the last two years. The average number of *commits* per day, registered since the project began is almost one thousand. The development assistance tools used by the GNOME project are basically the same as those used by KDE, and so we will not go into them in this section.

Table 10. Analysis of GNOME

Website	http://www.gnome.org
Beginning of the project	September 1997
License	GNU GPL and GNU LPGL
Analysed version	2,2

Source code lines.	9,200,000
Number of files (code, documentation, etc.)	228,000
Cost estimate	$ 400,000,000
Runtime estimate	11.08 years (133.02 months)
Estimate of average number of developers	Approximately 250
Number of subprojects	More than 700 modules in the CVS.
Approximate number of developers	Almost 1,000 with write-access to the CVS.
Number of *committers* active in the last year	Around 500 (approximately 55% of the total)
Number of *committers* active in the last two years	Approximately 700 (75% of the total)
Number of *commits* in the CVS	1,900,000
Average number of *commits* (total) per day	Approximately 900
Development assistance tools	CVS, mailing lists, website, news site, annual meetings

Whereas in KDE, C++ is undoubtedly the most widely-used language, in GNOME, the language is C. In GNOME, as occurs in KDE, this is due to the fact that the main library is written in C, which means that the *native* language is C, whereas programmers wishing to use the other languages have to wait for the corresponding bindings to appear. The most advanced language binding in GNOME is the one that is included in gnome--, which is none other than C++, which is why it is not surprising that that is the second language in the classification. Perl has always been widely accepted within the GNOME community and an example of this fact is that in GNOME it is possible to program in many languages. Its implementation, however, has not been as extensive as could have been expected and it is slightly more extensive than Shell. On another note, Python and Lisp were accepted fairly widely in GNOME, as is proven by the relative importance of this classification, whereas Java has never really taken off probably due to an incomplete link.

Table 11. Programming languages used in GNOME

Programming language	Code lines	Percentage
C	7,918,586	86.10%
C++	576,869	6.27%
Perl	199,448	2.17%
Shell	159,263	1.73%
Python	137,380	1.49%

Programming language	Code lines	Percentage
Lisp	88,546	0.96%

9.4.6. Academic studies on GNOME

The most important studies on GNOME in the academic sphere are the following two: "Results from software engineering research into open source development projects using public data" [158] and "The evolution of GNOME" [123].

- [158] is one of the first large-scale studies of software in the sphere of free software. The authors of the study took advantage of the fact that the details of the development are usually publicly accessible in order to measure the efforts and compare them against the cost estimate models, and traditional time and effort measurements. One of the classical models with which they compared them was the one used in this chapter, model COCOMO.

- [123] briefly goes over the objectives of GNOME and its short history, as well as the GNOME project's use of technology.

9.5. Apache

The HTTP Apache server is one of the star applications of the world of free software, as it is the web server that is most widely implemented, according to the Netcraft real-time survey (http://news.netcraft.com/archives/2003/08/01/august_2003_web_server_survey.html) [167]. For example, in May 1999, 57% of web servers worked with Apache, whereas in May 2003, the percentage had increased to 68%. Apache is available for all types of Unix (BSD, GNU/Linux, Solaris...), Microsoft Windows and other minority platforms.

9.5.1. History of Apache

In March 1989, Tim Berners Lee, an English scientist that worked in the CERN (Switzerland) proposed a new method for managing the huge amount of information from the CERN projects. The method would be a network of hyperlinked documents (hypertext, as Ted Nelson had called it already in 1965); the WWW was born. However, it was not until November 1990 that the first WWW software was unveiled: a package called the World Wide Web included a web browser with a graphic interface and a WYSIWYG ("what you see is what you get") editor. Two years later, the list of WWW servers had approximately thirty entries, including NCSA HTTPd.

The real history of Apache began when Rob Mc Cool left the NCSA in March 1995. Apache 0.2 would be born on 18th March 1995, based on the NCSA HTTPd 1.3 server, built by Rob McCool himself while he was at NCSA. During those first months, Apache was a collection of patches applied to the NCSA server, until Robert Thau launches Shambhala 0.1, an almost complete reimplementation that already included the API for the modules that subsequently turned out to be so successful.

Note

The name of the Apache project is based on its philosophy of development and organisation. As was the case with the Apache tribe, the developers of Apache decided that their organisational method should be based on the personal merits of the developers in comparison with the rest of the Apache community. However, there is a legend that has spread that says that the name Apache really came from the fact that in the initial stages, it was simply a patched NCSA server, or *a patchy server*.

The first stable version of Apache did not appear until January 1996, when Apache 1.0 was released, which included the loading of modules in test-mode runtime, as well as other interesting functions. The first months of that year were especially fruitful for the project, as version 1.1, which had authentication modules that would be checked against the databases (such as MySQL) was published only two moths later. From that time to today, the most important events for the project have been the introduction of total compliance with the HTTP 1.1 standard (included in April 1997 in Apache 1.2), the inclusion of the Windows NT platform (which began in July 1997 with the test versions of Apache 1.3), the unification of the configuration files in one single file (which did not happen until October 1998, in Apache 1.3.3) and the launch, still in the test stage, of the next generation of Apache, Apache 2.

In the meantime, in June 1998, IBM decided that, instead of developing its own HTTP, it would use Apache as the engine of its product WebSphere. This was interpreted as a huge endorsement for the Apache project from the Big Blue and for free software in general, although it would be necessary to modify the original Apache license slightly in order to make this work.

9.5.2. Development of Apache

The HTTP Apache server is the main project among the many that the Apache Software Foundation manages. The modular design of Apache has made it possible for there to be a series of satellite projects, based around Apache, some of which have even been bigger than Apache itself. For example, the HTTP Apache server contains the kernel of the system with the basic functionalities, whereas the additional functionalities are provided by different modules. The most well-known modules are *mod_perl* (a Perl script language interpreter embedded in the web server) and Jakarta (a powerful applications server). In the

following paragraphs, we will only describe the development process followed for the HTTP server, without taking into account the other modules, which may have similar modules or not.

The development of the HTTP Apache server is based on the work of a small group of developers called the Apache Group. The Apache Group consists of the developers that have worked together on the project for a long period of time, generally more than six months. The developer, having been invited by a member of the Apache Group to join, is voted in by all the other members. In the early stages, the Apache Group consisted of eight developers; this number then increased to twelve and there are currently twenty five members.

The Apache Group is responsible for the development of the web server and, therefore, for specific decisions regarding the development at any given moment. It is important to distinguish the Apache Group from the developers in the *core group*, which is active at all times. The voluntary nature of the work performed by most of the developers makes it unlikely that all the people that comprise the Apache Group will be active at all times, which means that the *core* is defined as the people who may take care of the tasks in Apache in a certain period of time. In general, the decisions that have to be made by the developers belonging to the core group are limited to voting for the inclusion or not of code, although in reality this is reserved only for large-scale changes and questions of design. On another note, they usually have write-access to the CVS repository, which means that they act as guardians for the incoming code, ensuring that it is correct and of good quality.

9.5.3. X-ray picture of Apache

The figures shown below correspond to the HTTP Apache server version that was available for download from the CVS server of the Apache project as at 18th April 2003. None of the numerous modules that the Apache project has, have been taken into account here. As we will see, the Apache project is relatively small as compared with the other cases studied in this chapter. Although this has already been mentioned, it is important to emphasise the modularity of Apache, which has the following specific advantages: the kernel is small and manageable. The CVS repository of the Apache project, which contains the kernel of the web server and many additional modules, hosts more than four million lines of source code, a figure that is slightly lower than those of projects such as KDE and GNOME.

Version 1.3 of Apache had a little more than 85,000 lines of source code; according to the COCOMO model, this would have required the work of an average of twenty developers working full-time for a year and a half. The total cost of the project would, at the time, be approximately 4 million dollars. In

order to prepare the Apache web server, up to sixty different *committers* would have been necessary, whereas the number of developers providing input, according to the calculations, would have been approximately four hundred.

Table 12. Analysis of Apache

Website	http://www.apache.org
Beginning of the project	1995
License	Apache Free Software License
Analysed version	2.2.4
Source code lines.	225,065
Number of files	2,807
Cost estimate	$ 7,971,958
Runtime estimate	2.52 years (30.27 months)
Estimate of average number of developers	23.4
Approximate number of developers	60 *commiters* (400 developers)
Development assistance tools	CVS, mailing lists, bug report system

Apache 1.3 is written almost completely in C language and there are scarcely any other programming languages, especially if we take into account the fact that most of the lines written in the second language, Shell, correspond to configuration files and compilation assistance.

Table 13. Programming languages used in Apache

Programming language	Code lines	Percentage
C	208,866	92.8%
Shell	12,796	5.69%
Perl	1,649	0.73%
Awk	874	0.39%

9.6. Mozilla

The Mozilla project works on a set of integrated applications for Internet, that are free and multiplatform, and the most notable products are the Mozilla Firefox web browser and the Mozilla Thunderbird email and news client. This set is also designed as a platform for developing other applications, which means that there are many browsers that use Gecko, Mozilla's HTML engine (such as Galeon).

The project is managed by the Mozilla Foundation, a non-profit organisation that creates free software and is "dedicated to preserving choice and promoting innovation on the Internet". For this reason, Mozilla's products are based on three basic principles: they must be free software, respect the standards and be portable to other platforms.

9.6.1. History of Mozilla

The history of Mozilla is long and convoluted but also very interesting, as it allows us to follow the history of the WWW itself. The reason for this is that if we trace all the persons and institutions that have been involved in the development of Mozilla, then we arrive at the starting point of the Internet, with the launch of the first complete Internet browser.

As was the case with Apache's predecessor, it was the NCSA where the first complete Internet browser, Mosaic, was "born" in 1993. Many of the members of the development team, with Marc Andreessen and Jim Clark at the helm, created a small company in order to write, starting from zero (as there were problems with the copyright on the code of Mosaic and the technical design of the program had its limitations, see *Speeding the Net: the inside story of Netscape and how it challenged Microsoft* [189]), what would subsequently become the Netscape Communicator browser, which was, unarguably, the leader of the market of Internet browsers until the arrival of Microsoft Internet Explorer. Apart from the purely technological innovation that the Netscape browser represented, Netscape Inc. was also innovative in the way it managed to corner the market. Completely contrary to what was held as common sense at the time, its star application, the WWW browser, was available for free (and could even be distributed with certain limitations). This approach, which was completely unheard of in the corporate world at the time, caused a certain amount of surprise, but it turned out to be right for Netscape Inc.'s strategy, and it was only the giant Microsoft that was able to outdo it with more aggressive (and probably detrimental to free market competition) tactics.

Around 1997, Netscape's market share had dropped sharply due to the spread of Microsoft Explorer; consequently, Netscape Inc. was studying new ways of recovering its previous dominance. A technical report published by the engineer Frank Hecker ("Setting up shop: the business of open source software", 1998) [142] proposed that the best solution to the problem was to release the source code of the browser and benefit from the effects of the free software community, as described by Eric Raymond in "The Cathedral and the Bazaar". In January 1998, Netscape Inc. officially announced that it would publicly release the source code of its browser, marking an extremely important milestone within the short history of free software: a company was going to publish the whole of the source code of an application that had been a commercial product up until then, under a free software license. The date of the launch was scheduled for the 31st March 1998.

In the two months between January and March, the people at Netscape were frenetically active, trying to get everything ready. The list of tasks was enormous and complicated ("Freeing the source: the story of Mozilla", 1999) [134]. On the technical level, it was necessary to contact the companies that made the modules to ask them for their consent to the change of license; if the answer was negative, the module had to be eliminated. In addition, all the parts written in Java had to be reimplemented, as it was considered that Java was not free. They then decided to call the free project Mozilla, just as the developers of Netscape had called their main component Netscape, and the Mozilla.org domain was purchased to build a community of developers and assistants based around this website. At the end of the process, more than one million and a half lines of source code were released.

Note

The name Mozilla is a play on words, with a little dose of humour from of the Netscape Inc. development team. The Mozilla name came from adapting the name Godzilla, the monster that caused mayhem in Japanese horror films from the fifties, to make it sound like *Mosaic Killer*, as the new browser, with more advanced technology, was supposed to render Mosaic obsolete.

On another note, there was the legal question. The free licenses existing at that time did not convince the Netscape executives, who could not see how this could be "compatible" with the commercial nature of a business. Netscape wanted a more *flexible* license, that would make it possible to reach agreements with third parties so as to include their code regardless of the license or whether other commercial developers were to contribute to it, so that they could defend their financial interests howsoever they chose. And although they had not initially planned to create a new license, they eventually reached the conclusion that this was the only way they could achieve what they wanted. This is how the Netscape Public License (NPL) was created: a license that was based on the basic principles of free software licenses, but that also gave certain additional rights to Netscape Inc, which also made it a *non-free* license, from the perspective of the Free Software Foundation. When the draft of the NPL was published for public discussion, the clause providing additional rights to Netscape was heavily criticised. Netscape Inc. reacted quickly in response to these criticisms and created an additional license, the Mozilla Public License (MPL), which was identical to the NPL, except in that Netscape had no additional rights.

The final decision was to release the Netscape code under the NPL license, which provided additional rights to Netscape, and that any new code that was included would be issued under the MPL (or a compatible license). The corrections to the original code (licensed with the NPL) would also be covered by this license.

Note

Currently, Mozilla accepts contributions under three licenses: the MPL, the GPL and the LGPL. Changing the license was not at all easy, as they had to find all the people that had contributed code at any point so that they would give their consent to the changeover from the NPL/MPL to the MPL/GPL/LGPL. In order to relicense the whole code, a website, which contained a list of three hundred "lost" hackers, was created ("Have you seen these hackers?") [38]. As at May 2007, they are still looking for two of these developers.

Developing the original code of Netscape Communicator was, without a doubt, more complicated than initially expected. The initial conditions were already bad to start with, because what was released was, on occasions, incomplete (all the third party modules for which no consent had been given for the release had been removed) and it hardly worked. As if that were not enough, apart from the technical problem of making Mozilla work on a large number of operating systems and platforms, there were the flaws taken from Netscape Inc., with release cycles that were too long and inefficient for the world of Internet and that did not distinguish between its own interests and the community formed around Mozilla. All of this came to a head exactly a year later when one of the most active programmers from before and after the release, Jamie Zawinsky, decided to throw in the towel in a bitter letter ("Resignation and post-mortem", 1999) [237] in which he made clear his despair and desolation.

On 15th July 2003, Netscape Inc. (now the property of America On Line) announced that it was no longer going to develop the Netscape browser and, therefore, was no longer going to actively take care of the Mozilla project. As a kind of "redundancy settlement" Netscape approved the creation of the Mozilla Foundation, which it supported with a contribution of two million dollars. Likewise, all of the code that was under the NPL (Netscape's public license) was donated to the Foundation and redistributed with the licenses previously published by the Mozilla project: MPL, LGPL and GPL.

On 10th March 2005, the Mozilla Foundation announced that it would not publish any more official versions of the Mozilla Application Suite, which would be replaced by Mozilla SeaMonkey, which would include a web browser, an email client, an address book, an HTML editor and an IRC client. On another note, the Mozilla project hosts various independent applications, the most notable of which include Mozilla Firefox (web browser), which is undoubtedly the most well-known, Mozilla Thunderbird (email and news client), Mozilla Sunbird (calendar), Mozilla Nvu (HTML editor), Camino (web browser designed for Mac OS X) and Bugzilla (web-based bug-tracker tool).

As time has progressed, despite the many doubts and the long periods in which it seemed that it was destined to fail, the project now seems to be going well. Thanks to the versatility and portability of its applications, despite requiring many runtime resources in many cases, they are used (generally, but especially Firefox) as the OpenOffice.org pair in the end user's desktop.

9.6.2. X-ray picture of Mozilla

The figures that we will discuss in this section correspond to a study of Firefox, the most well-known of the project's applications. According to the estimates of the COCOMO model, a company that wished to create software of this scale would have to invest approximately 111 million dollars to obtain it. The time it would take would be about seven years and the average number of programmers working full-time that the company would have to use would be approximately one hundred and twenty.

Table 14. Current status of Mozilla Firefox

Website	www.mozilla-europe.org/es/products/firefox/
Beginning of the project	2002
License	MPL/LGPL/GPL
Version	2.0
Source code lines.	2,768,223
Cost estimate	$ 111,161,078
Runtime estimate	6.87 years (82.39 months)
Estimate of average number of developers	120
Approximate number of developers	50 *committers*
Development assistance tools	CVS, mailing lists, IRC, Bugzilla.

C++ and C are the languages that are used the most, in that order of priority. Perl is used and this is mainly due to the fact that the development assistance tools created by the Mozilla project, such as BugZilla or Tinderbox, are designed in this language. What is surprising is the large amount of code lines in an assembly language in an end user application. An inspection of the code in the repository shows that, in effect, there are quite a lot of files encoded in assembly language.

Table 15. Programming languages used in Mozilla Firefox

Programming language	Code lines	Percentage
C++	1,777,764	64.22%
C	896,551	32.39%
Assembler	34,831	1.26%
Perl	26,768	0.97%
Shell	16,278	0.59%
C#	6,232	0.23%

Programming language	Code lines	Percentage
Java	5,352	0.19%
Python	3,077	0.11%
Pascal	459	0.02%

9.7. OpenOffice.org

OpenOffice.org is one of the star applications in the current free software scene. It is a multiplatform office application *suite* that includes the key applications in an office desktop environment, such as a word processor (Writer), a spreadsheet (Calc), a presentation program (Impress), a graphics editor (Draw), a tool for creating and editing mathematical formulae (Math) and, finally, an HTML language editor (included in Writer). The interface provided by OpenOffice.org is homogeneous and intuitive, with an appearance and functionalities similar to those of other office applications, especially the one that is most widely used today, Microsoft Office.

Written in C++, OpenOffice.org includes Java's API and has its own components for embedded systems, which makes it possible to include, for example, tables from a spreadsheet in the word processor in a very simple and intuitive way. One of its advantages is that it can handle a large amount of file formats, including those of Microsoft Office. Its native file formats, unlike those of Microsoft's office suite, are based on XML, which shows that they are clearly committed to versatility, the ease of transformation and transparency. Currently, OpenOffice.org has been translated into more than twenty five languages and it runs on Solaris (its native system), GNU/Linux and Windows. Versions for FreeBSD, IRIX and Mac OS X are expected in the not-too-distant future.

OpenOffice.org took its definitive name (OpenOffice, as everybody knows it, plus the *.org* tag) after a court case, in which it was accused of trademark violation by another company.

9.7.1. History of OpenOffice.org

In mid-1980s, the company StarDivision was founded in the Federal Republic of Germany, with the principal aim of creating an office application suite: StarOffice. In summer 1999, SUN Microsystems decided to purchase the company StarDivision and make a significant commitment to StarOffice, with the clear intention of wresting away part of the market share conquered by Microsoft at that time. In June 2000, the company launched version 5.2 of StarOffice, which could be downloaded for free by Internet.

However, StarOffice's success was limited, as the market was already strongly dominated by Microsoft's office package. SUN decided to change its strategy and, as occurred with Netscape and the Mozilla project, decided to take advantage of free software to gain importance and implement its systems. Consequently, the future versions of StarOffice (a proprietary product of SUN) would be created using OpenOffice.org (a free product) as a source, respecting the application programming interfaces (API) and the file formats and serving as the standard implementation.

9.7.2. Organisation of OpenOffice.org

OpenOffice.org aims to have a decision-making structure in which all the members of the community feel like participants. Consequently, a system was devised so that the decision-making process would have the greatest consensus possible. The OpenOffice.org project is divided into a series of subprojects that are taken on by project members, the assistants and one single leader. Of course, the members of a project may work on more than one project, as can the leader. However, no one can lead more than one project at a time. The projects are divided into three categories:

- Accepted projects. These can be technical or non-technical. The leaders of each accepted project have a vote when it comes to making global decisions.

- *Native-lang* projects. These are all the internationalisation and localisation projects of OpenOffice.org. Currently, as we have mentioned, there are more than twenty five teams that are working on translating the OpenOffice.org applications to different languages and conventions. As a set, *native-lang* projects have one single vote on global decisions.

- Incubating projects. These are the projects promoted by the community (generally, they are experimental or small). They may become accepted projects after a period of six months. In effect, the OpenOffice.org community can guarantee that the accepted projects are based on a real interest, as the *mortality rate* of new projects in the world of free software is very high. In total, the incubating projects have one vote on the decisions made.

9.7.3. X-ray picture of OpenOffice.org

The OpenOffice.org office suite comprises approximately four million lines of source code distributed throughout forty five thousand files.

The COCOMO model estimates that the work required to build a "clone" of OpenOffice.org would have to be provided by one hundred and eighty programmers working full-time for almost eight years. According to the COCOMO estimates, the development cost would be approximately 215 million dollars.

The results discussed in this section were obtained from a study of the source code of stable version 2.1 of OpenOffice.org.

Table 16. Current status of OpenOffice.org

Website	http://www.openoffice.org
Beginning of the project	June 2000 (first free versions)
License	LGPL and SISSL
Version	2.1
Source code lines.	5,197,090
Cost estimate	$ 215,372,314
Runtime estimate	8.83 years (105.93 months)
Estimate of average number of developers	180
Approximate number of developers	200 *commiters*
Development assistance tools	CVS, mailing lists

Where the programming languages used in OpenOffice.org are concerned, the most prevalent is C++. It is interesting to note how Sun's purchase of the company resulted in the integration of a lot of Java code in the office suite, which even exceeded the amount of language in C.

Table 17. Programming languages used in OpenOffice.org

Programming language	Code lines	Percentage
C++	4,615,623	88.81%
Java	385,075	7.41%
C	105,691	2.03%
Perl	54,063	1.04%
Shell	12,732	0.24%
Yacc	6,828	0.13%
C#	6,594	0.13%

9.8. Red Hat Linux

Red Hat Linux was one of the first commercial distributions of GNU/Linux. Today, it is probably one of the most well-known, and certainly the one that can be considered the "canonical" of all the commercial distributions. The work of the distributors is basically related to integration tasks and not so much to software development. Of course, Red Hat and other distributions may have developers working for them, but their work is secondary for the aims of a distribution. In general, it is assumed that the task of the distributions is to simply take the source packages (generally the files published by the developers themselves) and bundle them so that they fulfil certain criteria (both technical and organisational). The product of this process is a distribution: a series of properly organised bundles that make it possible for the user to install, uninstall and update them.

Distributions are also responsible for the quality of the final product, which is a very important aspect if we consider that many of the applications that are included have been developed by volunteers in their free time. Consequently, the security and stability aspects are of the essence for a distribution.

9.8.1. History of Red Hat

Red Hat Software Inc. was founded by Bob Young and Marc Ewing in 1994. The main objective was to compile and commercialise a GNU/Linux distribution that was called (and is still called) Red Hat Linux [236]. Basically, it was a bundled version of what existed on the Internet at that time, including documentation and support. Version 1.0 of this distribution was born in the summer of 1995. A few months later, in autumn, version 2.0, which included RPM technology (*RPM package manager* was published. The RPM package manager has become a *de facto* standard for bundles in GNU/Linux systems. In 1998, version 5.2 of Red Hat was issued to the great public. For a complete history of the names of the different versions of Red Hat, please read "The truth behind Red Hat names" [201].

Note

As of version 1.1 of Linux Standard Base (a specification designed to achieve binary compatibility between GNU/Linux distributions, which is taken care of by the Free Standards Group), RPM has been chosen as the standard package manager. The Debian project continues with its own package format, as do many distributions that depend on the Debian package management system, and they are adjusted to the standardised format using a conversion tool called *alien*.

Before the RPM management system existed, almost all of the GNU/Linux distributions offered the possibility of installing the software through a menu-based procedure, but making modifications to an existing installation, especially adding new software packages after the installation, was not easy. RPM made that step beyond the state-of-the-art possible by providing users with the ability to manage their own packages ("Maximum RPM. Taking the Red Hat package manager to the limit", 1998) [83], which made it possible to delete,

install or update any software package existing in the distribution in a much easier way. The RPM package system continues to be the most widely used package management system in the different GNU/Linux distributions. The statistics of Linux Distributions, "Facts and figures", 2003 [92], a website that contains qualitative and quantitative information on a large number of distributions, show that in May 2003, a large majority (sixty five) of the one hundred and eighteen distributions used for the calculations, used the RPM (approximately 55% of the total). In comparison, the Debian package format (known as *deb*) was only used in sixteen distributions (approximately 14% of the total).

However, Red Hat Inc. was not only known for its software distribution based on Linux. In August 1999, Red Hat went public and its shares achieved the eighth highest first day gain in the whole of Wall Street history. Four years later, the value of Red Hat's shares had shrunk to a hundredth of the maximum value they reached before the dotcom crisis. Nevertheless, its successful beginnings on the stock market put Red Hat on the front pages of newspapers and magazines that did not specialise directly in IT matters. In any case, it seems that Red Hat has managed to overcome the problems that other companies in the business world have had with free software and the numbers it published in the last quarter of 2002 were in the black for the first time in its history.

Another of the most important historical events involving Red Hat was the acquisition of Cygnus Solutions in November 1999, a company founded a decade before that had already proved how it was possible to earn money with an integral strategy based on free software ("Future of Cygnus Solutions. An entrepreneur's account") [216]. Cygnus chose the complicated market of compilers to make its mark. Its commercial strategy was based on the development and adaptation of GNU software development tools (basically GCC and GDB) tailored to the client's needs.

In September 2003, Red Hat decided to concentrate its development work on the corporate version of its distribution and delegated the common version to Fedora Core, an open source project independent of Red Hat.

In June 2006, Red Hat purchased the company JBoss, Inc., becoming the company in charge of developing the most important open source applications server, J2EE.

9.8.2. Current status of Red Hat.

Currently, Red Hat Inc.'s most important products are Fedora Core and Red Hat Network, an Internet software update service. These types of services are designed more with the end user in mind and not so much for the corporate environment, but they are good for Red Hat to advertise itself and to reinforce its brand strategy.

Red Hat's "real" commercial strategy is based on the products it designs for the corporate world. These types of products are much less well-known, but they constitute a major part of Red Hat's turnover, much greater than that of its most popular star products in the literal sense.

Red Hat has a distribution that is corporate-orientated, integrated around an applications server called Red Hat Enterprise Linux AS. Clients that purchase this software also receive support. The equivalent of Red Hat Network for commercial users is Red Hat Enterprise Network, which includes system management and the option of obtaining updates. On anther note, Red Hat also offers IT consultancy services and a certification program similar to that offered by Microsoft in the world of Windows.

9.8.3. X-ray of Red Hat

Red Hat has recently passed the milestone of fifty million lines of code, which makes it one of the biggest software distributions that have ever existed, exceeding, as we shall see later in this chapter, the size of proprietary operating systems. Red Hat Version 8.1 consisted of 792 packages, so we can assume that the latest version would have had more than eight hundred packages, if we consider that the number tends to increase slightly from version to version.

As in our previous examples, the COCOMO model has been used to estimate the investment and effort that would have been necessary in order to create a generation of software of the same scale. However, in Red Hat's case, we have taken into account the fact that it is a product prepared using a series of independent applications. Consequently, an independent COCOMO estimate has been used for each one of Red Hat's packages, and then we have added the estimated costs and personnel that would have been required. In order to analyse the optimum design time for Red Hat, we have chosen the largest package, as, in theory, all the packages are independent and could therefore be designed at the same time. For this reason, the optimum design time for Red Hat is similar to that of the other projects presented in earlier sections of this chapter.

According to COCOMO, approximately seven and a half years and a team of developers, consisting of an average of one thousand eight hundred developers, would have been necessary in order to design the Red Hat Linux 8.1 distribution, starting from zero. The cost of the final development would be approximately 1,800 million dollars.

Note

One thousand eight hundred million dollars is the sum that the Spanish Ministry of Defence has allocated to renewing its helicopter fleet in the latest budget. Out of that sum, half will be invested in buying twenty four helicopters, so we could say that the price of Red Hat would be equivalent to that of forty eight combat helicopters. Likewise, 1,800 million dollars is the total global earning from the film *Titanic*.

Table 18. Status of Red Hat Linux.

Website	http://www.redhat.com
Beginning of the project	1993
License	
Version	9.0
Source code lines.	More than 50,000,000
Number of packages	792
Cost estimate	$ 1,800,000,000
Runtime estimate	7.35 years (88.25 months)
Estimate of average number of developers	1,800
Approximate number of developers	Red Hat employees (generally integration only)
Development assistance tools	CVS, mailing lists

Due to the fact that there are many packages, the languages in Red Hat are more diverse than the ones we have seen in the most important free software applications. In general terms, C is very important, with more than sixty per cent of the code lines. In second place, with more than ten million lines of code, we have C++, followed by a long distance by Shell. It is interesting to note that after Perl we have Lisp (mainly due to its use in Emacs), assembly language (of which a quarter corresponds to the language that comes with Linux) and a language whose use is frankly declining, Fortran.

Table 19. Programming languages used in Red Hat.

Programming language	Code lines	Percentage
C	30,993,778	62.13%
C++	10,216,270	20.48%
Shell	3,251,493	6.52%
Perl	1,106,082	2.22%
Lisp	958,037	1.92%
Assembler	641,350	1.29%
Fortran	532,629	1.07%

9.9. Debian GNU/Linux

Debian is a free software operating system that currently uses the Linux kernel for its distribution (although it is expected that there will be Debian distributions based on other kernels in the future, as is the case of "the HURD"). It is currently (in 1997) available for various different architectures, including Intel x86, ARM, Motorola, 680x0, PowerPC, Alpha and SPARC.

Debian is not only the largest existing GNU/Linux distribution, but also one of the most stable and it has received various awards for the fact that it is preferred by users. Although its user base is difficult to estimate, as the Debian project does not sell CDs or any other media with its software and the software that it does have can be redistributed by anyone that wishes to, we can suppose, with a reasonable degree of certainty, that it is an important distribution within the GNU/Linux market.

There is a categorisation in Debian that depends on the license and distribution requirements of the packages. The kernel of the Debian distribution (the section called *main* that covers a great variety of packages) consists only of free software in accordance with the DFSG (Debian Free Software Guidelines) [104]. It can be downloaded from the Internet and many redistributors sell it on CDs or in other media.

Debian distributions are created by almost one thousand volunteers (generally IT professionals and experts). The work of these volunteers consists of taking the source programs, in most cases from the original authors, configuring them, compiling them and bundling them so that an average Debian distribution user only has to select the package and the system will install it with no further problems. What may first appear as simple can become complex as soon as other factors, such as the dependencies between the different packages (package A needs package B in order to work) and the different versions of all these packages, are taken into account.

The work performed by the members of the Debian project is the same as that performed in any other distribution: the integration of the software so that it all works together properly. Apart from the adaptation and bundling work, the Debian developers are in charge of maintaining an Internet-based services infrastructure (website, online files, bug management system, assistance mailing lists, support and development, etc.), various translation and internationalisation projects, the development of various tools specific to Debian and, generally, in charge of anything that is required in order to make the Debian distribution work.

Apart from its voluntary nature, the Debian project has a feature that is especially unique: Debian's social contract (http://www.debian.org/social_contract.html) [106]. This document does not only describe the Debian project's main goals, but also the means that will be used to achieve them.

Debian is also known for having a very strict packages and versions policy, designed to achieve the best quality in the product (the "Debian policy manual") [105]. In this way, there are three different *types* of Debian at any given time: a stable version, an unstable version and a test version. As the name itself indicates, the stable version is the one recommended for systems and users that require complete stability. The software has to be subjected to a freeze period, during which any bugs are corrected. The general rule is that the stable Debian version must not have any known critical bug. On the other hand, this stable version does not usually have the latest versions of the software (the newest additions).

There are another two versions of Debian that exist alongside the stable one for those that wish to have the most recent software. The unstable version includes packages that are being stabilised, whereas the test version, as the name indicates, is the one that has a greater tendency to fail and that contains the newest of the new in terms of the latest software.

When the first study was performed, the stable version of Debian was Debian 3.0 (also known as Woody), the unstable one was nicknamed Sid and the test version was Sarge. However, Woody also went through an unstable stage and, before that, a test stage. This is important, because what we will consider in this article will comprise the different stable versions of Debian, ever since version 2.0 was published, in 1998. For example, we have Debian 2.0 (alias Hamm), Debian 2.1 (Slink), Debian 2.2 (Potato) and, finally, Debian 3.0 (Woody).

Note

The nicknames of the Debian versions correspond to the main characters in the animated film *Toy story*, a tradition that started, half-jokingly and half-seriously, when version 2.0 was published and Bruce Perens, the leader of the project at the time and subsequent founder of the Open Source Initiative and the phrase *open source*, was working for the company that was designing the film. For more details regarding Debian's history and the Debian distribution in general, we recommend "A brief history of Debian" [122].

9.9.1. X-ray picture of Debian

Debian GNU/Linux is probably the largest compilation of free software that works in a coordinated manner and, doubtlessly, one of the biggest software products ever built. Version 4.0, released in April 2007 (called Etch), consists of more than ten thousand source packages, with more than 288 million lines of code.

The number of lines of code in Debian 3.0 is 105 million. According to the COCOMO model, a sum of approximately 3,600 million dollars would have to be paid in order to obtain software similar to that bundled with this distribution. As with Red Hat, the effort required to build each package separately has been calculated and the resulting figures have then been added to each other. For the same reason, the time it would have taken to develop Debian is only seven years, as the packages could have all been built at the same time as each other. However, an average of approximately four thousand developers would have had to have been mobilised during those seven years.

> **Note**
>
> Three thousand six hundred million dollars is the budget allocated by the 6th EC Framework Programme for research and development on the information society. It is also the sum that Telefónica intends to invest in Germany in order to deploy UMTS services.

Table 20. Status of Debian

Website	http://www.debian.org
Beginning of the project	16/08/1993
License	Those that fulfil the DFSG
Version used	Debian 4.0 (alias Etch)
Source code lines.	288,500,000
Number of packages	10,106
Cost estimate	$ 10,140 million
Runtime estimate	8.84 years
Approximate number of *maintainers*	Approximately 1,500
Development assistance tools	Mailing lists, bug report system

The most commonly used language in Debian 4.0 is C, with more than 51% of the lines of code. However, as we shall show a little later in this section, the importance of C is declining with time, as 80% of the code in the first versions of Debian, was in C. The second most commonly used language, C++, shares a fair part of the "blame" for the decline of C; however, C has especially been affected by *scripting languages* such as Perl, Python and PHP. Amongst these, languages such as Lisp or Java (which is underrepresented in Debian due to its policy of not accepting code that depends on Sun's proprietary virtual machine) sometimes manage to get in.

Table 21. Programming languages used in Debian GNU/Linux 4.0

Programming language	Lines of code (in millions)	Percentage
C	155	51%
C++	55	19%
Shell	30	10%
Perl	8.1	2.9%
Lisp	7.7	2.7%

Programming language	Lines of code (in millions)	Percentage
Python	7.2	2.5%
Java	6.9	2.4%
PHP	3.5	1.24%

Table 22 shows how the most important languages developed in Debian.

Table 22. Languages most used in Debian

Language	Debian 2.0		Debian 2.1		Debian 2.2		Debian 3.0	
C	19,400,000	76.67%	27,800,00	74.89%	40,900,000	69.12%	66,500,000	63.08%
C++	1,600,000	6.16%	2,800,000	7.57%	5,980,000	10.11%	13,000,000	12.39%
Shell	645,000	2.55%	1,150,000	3.10%	2,710,000	4.59%	8,635,000	8.19%
Lisp	1,425,000	5.64%	1,890,000	5.10%	3,200,000	5.41%	4,090,000	3.87%
Perl	425,000	1.68%	774,000	2.09%	1,395,000	2.36%	3,199,000	3.03%
Fortran	494,000	1.96%	735,000	1.98%	1,182,000	1.99%	1,939,000	1.84%
Python	122,000	0.48%	211,000	0.57%	349,000	0.59%	1,459,000	1.38%
Tcl	311,000	1.23%	458,000	1.24%	557,000	0.94%	1,081,000	1.02%

There are languages that we could consider to be in the minority that reach fairly high positions in the classification. This is due to the fact that, whilst they are only present in a small number of packages, the packages in question are quite big. Such is the case of Ada, which whilst only being in three packages (GNAT, an Ada compiler; libgtkada, a link to the GTK library and ASIS, a system for managing Ada sources) covers 430,000 of the total 576,000 lines of source code that have been counted in Debian 3.0 for Ada. Another similar case is Lisp, which only appears in GNU Emacs and XEmacs, but has more than 1,200,000 lines of the approximately four million in the whole distribution.

9.9.2. Comparison with other operating systems

There is the proverb that says that all comparisons are odious; this is especially true when comparing free software with proprietary software. The detailed x-ray pictures taken of Red Hat Linux and Debian were possible because they are examples of free software. Having access to the code (and to the other information that has been provided in this chapter) is essential for studying the different versions' number of lines, packages, programming languages, etc. But the advantages of free software go beyond this, because, in addition, they make it easier for third parties, whether they are research teams or simply people that are interested, to analyse them.

In proprietary systems in general, a study such as this would be completely impossible. In fact, the figures provided below were obtained from the companies behind proprietary software development themselves, which means that we are not in a position to guarantee their truthfulness. To top this off, in many cases we do not know whether they are talking about physical source code lines, as we have done during this chapter, or whether they also include the blank lines and comments. Furthermore, neither do we know for certain what they include in their software, which means that we do not know whether certain versions of Microsoft Windows include the Microsoft Office suite or not.

In any case, considering all that we have discussed on this matter in previous paragraphs, we believe that it is interesting to include this comparison, as it helps us to see the position in which the different Red Hat and Debian distributions are in, within a wider context. What is unquestionable is that both Debian and Red Hat, but especially the former, are the largest collections of software ever seen by humanity to date.

The figures cited below come from Mark Lucovsky [168] for Windows 2000, SUN Microsystems [171] for StarOffice 5.2, Gary McGraw [169] for Windows XP and Bruce Schneier [200] for all the other systems. Table 23 provides a comparison, from smallest to greatest.

Table 23. Comparison with proprietary systems

System	Date of publication	Lines of code (approx.)
Microsoft Windows 3.1	April 1992	3,000,000
SUN Solaris 7	October 1998	7,500,000
SUN StarOffice 5.2	June 2000	7,600,000
Microsoft Windows 95	August 1995	15,000,000
Red Hat Linux 6.2	March 2000	18,000,000
Debian 2.0	July 1998	25,000,000
Microsoft Windows 2000	February 2000	29,000,000
Red Hat Linux 7.1	April 2001	32,000,000
Debian 2.1	March 1999	37,000,000
Windows NT 4.0	July 1996	40,000,000
Red Hat Linux 8.0	September 2002	50,000,000
Debian 2.2	August 2000	55,000,000
Debian 3.0	July 2002	105,000,000

9.10. Eclipse

The Eclipse platform consists of an open and extensible IDE (*integrated development environment*). An IDE is a program consisting of a set of tools that are useful for a software developer. The basic elements of an IDE include a code editor, a compiler/interpreter and a debugger. Eclipse is an IDE in Java and provides numerous software development tools. It also supports other programming languages, such as C/C++, Cobol, Fortran, PHP or Python. *Plug-ins* can be added to the basic platform of Eclipse to increase the functionality.

The term Eclipse also refers to the free software community that develops the Eclipse platform. This work is divided into projects with the aim of providing a robust, scalable and quality platform for the development of software with the Eclipse IDE. The work is coordinated by the Eclipse Foundation, which is a non-profit organisation created for the promotion and development of the Eclipse platform and that supports both the community and the Eclipse ecosystem.

9.10.1. History of Eclipse

A lot of Eclipse's programming was carried out by IBM before the Eclipse project was created as such. Eclipse's predecessor was VisualAge and it was built using Smalltalk in a development environment called Envy. After Java appeared in the nineties, IBM developed a virtual machine that worked with both Smalltalk and Java. The rapid growth of Java and its advantages with the focus on an Internet that was expanding heavily forced IBM to consider abandoning this dual virtual machine and to build a new platform based on Java from scratch. The final product was Eclipse, which had already cost IBM approximately 40 million dollars in 2001.

Towards the end of 2001, IBM, along with Borland, created the non-profit Eclipse foundation, thereby opening up to the open source world. This consortium was gradually joined by important global software development companies: Oracle, Rational Software, Red Hat, SuSE, HP, Serena, Ericsson and Novell, among others. There are two significant absences: Microsoft and Sun Microsystems. Microsoft was excluded due to its monopoly of the market and Sun Microsystems had its own IDE, constituting Eclipse's main competition: NetBeans. In fact, the Eclipse name was chosen because the aim was to create an IDE able to "eclipse Visual Studio" (Microsoft) and to "eclipse the sun" (Sun Microsystems).

The latest stable version of Eclipse is available for the Windows, Linux, Solaris, AIX, HP-UX and Mac OS X operating systems. All versions of Eclipse need to have a Java Virtual Machine (JVM) installed in the system, preferably JRE (Java Runtime Environment) or JDK (Java Developer Kit) by Sun, which, as at early 2007, are not yet free (although Sun has announced that their JVM will be).

9.10.2. Current state of Eclipse

All the work prepared for the Eclipse consortium is organised into different projects. These projects are in turn divided into subprojects and the subprojects into components. The high-level projects are managed by committees of the Eclipse Foundation (PMC, *project management committees*). The following list shows the high-level projects:

- Eclipse. Base platform for the rest of the components. This platform will be free, robust, complete and of a good quality for the development of *rich client platforms* (RCP) and integrated tools (*plug-ins*). The Eclipse platform's runtime kernel is called Equinox and it is an implementation of the OS-Gi specification (Open Services Gateway Initiative), which describes a services oriented architecture (SOA) for applications.

- Tools (ETP, *Eclipse tools project*). Various tools and common components for the Eclipse platform.

- Web (WTP, *web tools project*). Tools for the development of web applications and JEE (Java Enterprise Edition).

- *Test and performance tools project* (TPTP). Testing tools and performance level measurers so that the developers can monitor their applications and make them more productive.

- Web reports (BIRT, *business intelligence and reporting tools*). Web report generation system.

- Modelling (EMP, *Eclipse modelling project*). Model-based development tools.

- Data (DTP, *data tools platform*). Support for data-handling technologies.

- Embedded devices (DSDP, *device software development platform*). Tools for the development of applications that are to be run on devices with limited hardware, in other words, embedded devices.

- *Service oriented architecture* (SOA). Tools for developing service-oriented projects.

- Eclipse Technology. Research, dissemination and development of the Eclipse platform.

The principles that guide the development of the Eclipse community are as follows:

- Quality. The software developed at Eclipse must meet the software engineering quality standards.

- Development. The Eclipse platform, and all the tools based on it, must develop dynamically in accordance with the users' requirements.

- Meritocracy. The more someone contributes, the more responsibilities they have.

- Eclipse Ecosystem. There will be resources donated by the open source community to the Eclipse consortium. These resources will be employed in ways that benefit the community.

Eclipse's development process follows certain predefined phases. Firstly, there is a phase called the pre-proposal phase, in which an individual or company declares their interest in establishing a project. If the proposal is accepted, it is decided whether it will be a high-level project or a subproject. The next step is to validate the project in terms of applicability and quality. After a phase in which the project is incubated, there will be a final revision. If the project passes this revision, it will have proved its validity before the Eclipse community and it will pass into the implementation phase.

9.10.3. X-ray of Eclipse

Eclipse is distributed under an EPL License (Eclipse Public License). This license is considered free by the FSF and the OSI. Under the EPL License, it is possible to use, modify, copy and distribute new versions of the licensed product. EPL's predecessor is the CPL (Common Public License). The CPL was written by IBM, whereas the EPL is the work of the Eclipse consortium.

Estimating the investment and effort put into Eclipse is not an easy task. This is due to the fact that the source code that comprises the Eclipse ecosystem is distributed in numerous projects and software repositories.

Below are the results of applying the COCOMO model to the Eclipse platform, which is used as the base for the rest of the *plug-ins*.

Table 24. Analysis of Eclipse

Website	http://www.eclipse.org
Beginning of the project	2001
License	Eclipse Public License
Analysed version	3.2.2
Source code lines.	2,163,932
Number of files	15,426

Cost estimate	$ 85,831,641
Runtime estimate	6.22 years (74.68 months)
Estimate of average number of developers	102.10
Approximate number of developers	133 *commiters*
Development assistance tools	CVS, mailing lists, bug-tracking system (Bugzilla)

The following table shows the programming languages used in Eclipse 3.2.2:

Table 25. Programming languages used in Eclipse

Programming language	Code lines	Percentage
Java	2,066,631	95.50%
C	85,829	3.97%
Perl	3,224	0.06%
C++	5,442	0.25%
JSP	3,786	0.17%
Perl	1,325	0.06%
Lex	1,510	0.03%
Shell	849	0.04%
Python	46	0.00%
PHP	24	0.00%

10. Other free resources

"If you want to make an apple pie from scratch, you must first create the universe."

Carl Sagan

Can the ideas behind free programs be extended to other resources? We could consider that other information resources that can easily be copied electronically are similar to programs and that the same freedoms, rules, development and business models could apply to them. However there are some differences and the implications of these differences have meant that they have not developed with the same force as programs. The main difference is that all one has to do is copy the programs to make them work, whereas when other types of information are copied, they have to pass through a more or less costly process before they can begin to be useful in any way, which can go from learning a document to the production phase of hardware described in the appropriate language.

10.1. The most important free resources

We already discussed the documentation of programs and other technical documents in section 3.2.5. Here we will look at other types of creations, which can also be textual, but which are not related to software, but rather to scientific, technical and artistic fields.

10.1.1. Scientific papers

The way in which science evolves is, to a large extent, due to the fact that the researchers that make it progress for the benefit of humanity publish the results of their work in journals that reach a wide public. Thanks to this dissemination, researchers develop a track record that allows them to progress towards positions of higher standing and responsibility, whilst they receive income from research contracts that they obtain thanks to their developing prestige.

This way of disseminating papers represents a *business model* that has proved very fruitful. For this model to work, the quality of the work has to be guaranteed and the papers must be widely disseminated. The obstacle that prevents the dissemination is the large amount of existing journals, of a significant cost, which can only be purchased with generous budgets. The quality is guaranteed by the fact that the papers are reviewed by specialists or peers.

In relation to this, numerous online journals have emerged, among which we would mention the veteran *First Monday* ("*First Monday*: peer reviewed journal on the Internet") [26] or the *Public Library Of Science* project (PLOS

http://www.publiclibraryofscience.org [55]). The "Directory of Open Access Journals" [22] cites many more. Should persons other than the authors be allowed to publish modifications to these types of papers? There are objections that range from the possibility of substandard quality or equivocation of opinions or results, to the danger of people that can easily plagiarise the papers and rise in the ranks with no effort, whilst denying the true authors of their hard-earned merits. However, the fact that all writers are under the obligation of citing the original author and of submitting the paper to a peer-review for publication in a prestigious journal can offset these problems (*see* section 10.2.2).

An analogy has been established between free software and science, as the development model of the former requires the greatest amount of dissemination, peer-reviews (presumably experts) and the reuse of results ("Free software/free science", 2001) [154].

10.1.2. Laws and standards.

There are documents of a regulatory nature that define how things must be done, so as to improve coexistence between people or so that programs or machines can operate together. These documents need to be widely disseminated, which means that any obstacles will be counterproductive. For this reason, it is understandable that they receive special treatment, as exemplified in the Spanish Intellectual Property Act:

> "Legal or regulatory provisions and drafts thereof, judgements of jurisdictional bodies and acts, agreements, deliberations and rulings of public bodies, and official translations of all such texts, shall not be the subject of intellectual property".

The technological equivalent of these laws would be the norms or standards. In programming, the communications protocols, either between remote machines or between modules in the same machine, are especially important. It is obvious that we should not limit their dissemination, especially if we want free programs that operate with others to flourish, but, despite this, traditionally, the bodies that regulate these matters, such as ISO[11] and ITU[12], sell their regulations and standards, even in electronic formats, and prohibit their redistribution. Although this can be justified to an extent, claiming the need to cover part of the costs, the free dissemination of the standards has been much more productive; this is the case of the W3C[13] guidelines and, especially where Internet standards are concerned, the documents called RFCs (*request for comments*) that have existed since the beginning, in electronic formats that can be read using any form of text editor.

[11]International Organisation for Standardisation

[12]International Telecommunications Union

[13]World Wide Web Consortium

However, the success of the Internet protocols is not due solely to their availability. Other factors include the *development model*, which is very similar to free software due to its openness to the participation of any interested person

and the use of mailing lists and similar elements. This process is described in "The Internet standards process - revision 3" [94] and "The Tao of IETF: A Novice's Guide to the Internet Engineering Task Force" [136].

Should modifying the texts of laws and regulations be allowed? Obviously not if it leads to confusion. For example, an RFC should only be modified in order to explain it or add clarifying comments, whereas not even this is allowed without an explicit authorisation for the recommendations of the W3C (http://www.w3.org/Consortium/Legal/2002/copyright-documents-20021231) [65]. The licenses themselves are also legal documents that cannot be modified. Should it be possible to create new regulations derived from other existing ones using the original documents? This would probably lead to the effortless spread of similar and incompatible regulations that would create confusion and could help the companies that dominate the market to promote their own incompatible variations, as it is in fact occurring, especially in the sphere of the Internet. Nevertheless, where State legislation is concerned, very often the laws have been copied literally from those of other countries and adapted with small modifications to the local particularities.

Is there a business model for laws and regulations? There are numerous professionals that work on the laws, in charge of designing, interpreting and enforcing them (legislators, lawyers, solicitors, judges, etc). There are laboratories that provide compliance certificates for the regulations. The regulatory bodies subsist, or should subsist, on the contributions of their members who wish to promote standards, for example, because their business is based on products that interoperate.

In the same way that it is convenient to have a definition of *free software* or *open software*, it is also necessary to have a working definition of *open standards*. Bruce Perens (http://perens.org/OpenStandards) [15] proposed the following definition based on the following principles:

1) Availability: if possible, open standards must be available for all to read and implement.

2) Maximise end user choice.

3) Open standards must be free for all to implement with no royalty or fee (certifications of compliance may involve a fee, although Bruce Perens advises that there should be free *self-certification* tools available).

4) No discrimination to favour one implementer over another.

5) Extension or subset permissions (non-certifiable).

6) Avoidance of predatory practices by dominant manufacturers. All proprietary extensions must have an open standard implementation.

10.1.3. Encyclopaedias

In 1999, Richard Stallman proposed the idea of a free encyclopaedia ("The free universal encyclopaedia and learning resource", 2001) [210] as a mechanism for avoiding the appropriation of knowledge and providing universal access to learning and the associated documents. It would consist of articles provided by the community, with no centralised control, where different actors would undertake different tasks, including, as a recommendation but not an obligation, that of revising or checking the articles. This encyclopaedia would not only contain text but also multimedia and free educational software.

Various initiatives have emerged to make this a reality. For instance, Nupedia (http://www.nupedia.com) [178] tried to build a quality encyclopaedia, but the attempt failed, perhaps because it required a format that was relatively difficult to learn (TEI), although probably more because of the requirement of having all the articles edited, revised by scientists and checked for style, etc.

The successor to Nupedia, which was much more successful, was Wikipedia (http://www.wikipedia.org) [69]. Wikipedia is a free multilingual encyclopaedia based on *wiki* technology. Wikipedia is written cooperatively by volunteers and the vast majority of articles can be modified by anyone with a web browser. Its success is based on its structure, which is more flexible in terms of editing, which eliminates the obstacles that Nupedia had in place and which makes it closer to what Stallman had in mind. The word *wiki* comes from the Hawaiian *wiki wiki* ('quick). *Wiki* technology allows anyone to edit any document using the structured text system, which is extraordinarily simple as we saw in section 8.6.2. In February 2007, the number of articles in English in Wikipedia was more than 1,500,000.

Note

Wikipedia is a project by the non-profit organisation Wikimedia, which also has the following projects, based on the same model as Wikipedia:

*   Wiktionary (http://www.wiktionary.org) [66]. This is a cooperative project that aims to create a free multilingual dictionary, with definitions, etymologies and pronunciations, in the required languages.

*   Wikibooks (http://www.wikibooks.org/) [67]. This is a project that aims to provide textbooks, manuals, tutorials or other pedagogic texts to anyone requiring these elements, for free.

*   Wikiquote (http://www.wikiquote.org) [70]. It is a compilation of famous phrases in all languages, which includes the sources when these are known.

*   Wikisource. It is a library of original texts that are in the public domain or that have been published with a GFDL (GNU free documentation license).

*   Wikispecies (http://species.wikimedia.org/) [71]. It is an open repertory of animal species, vegetable species, fungi, bacteria and all forms of known life.

*   Wikinews (http://wikinews.org/) [68]. It is a source of free news content in which the users are the editors.

*   Commons (http://commons.wikimedia.org/) [19]. It is a free repository of images and multimedia content.

*   Wikiversity (http://wikiversity.org/) [72]. It is an open and free educational platform, based on teaching projects at all educational levels.

*   Meta-Wiki (http://meta.wikimedia.org/) [48]. It is the website that supports all the projects of the Wikimedia Foundation.

We should also mention the *Concise Encyclopedia of Mathematics*, which has a more limited concept of what free means (it can only be consulted on Internet) and a development model in which it is necessary to submit all contributions to an editorial committee before publication.

10.1.4. Courses

With the same aim as the encyclopaedias, it is possible to produce free teaching materials, such as notes, slides, exercises, books, syllabic or didactic software. There is a tendency to view universities as businesses that produce and sell knowledge, which contradicts their basic principles. The reasons why a university may make these materials available to all are as follows:

*   The fulfilment of its mission, as an agent that disseminates knowledge.

*   The low cost of making existing materials available worldwide.

*   The fact that these materials cannot replace teaching in person.

*   The idea of these materials as publicity that may attract students and contribute to the university's prestige.

- The possibility of creating a community of teachers that review and improve the materials.

The most prominent initiative in this area is that of the MIT (http://ocw.mit.edu) [174], which has the aim of making more than two thousand well-catalogued resources accessible in a coherent and uniform manner.

10.1.5. Collections and databases

The mere compilation of information following determined criteria, organising it and making it available is, in itself, a product of valuable information, regardless of the information itself, which is therefore the product of its authors and, consequently, subject to restrictions on the freedom to access, modify or redistribute the content. Therefore, if we want free information, we can also want free collections.

For example, we may wish to classify important information in the Internet, organising and commenting the links. This is what the ODP (Open Directory Project http://dmoz.org [109]) does; it is operated by Netscape and is maintained by voluntary editors organised according to a hierarchical structure. The full directory can be freely copied in RDF format and published with certain modifications, as does Google and many other search engines that take advantage of it. Netscape, which owns the directory, guarantees an "Open Directory Project social contract"[53] inspired on that of the Debian distribution (http://www.debian.org/social_contract.html) [106], which facilitates external contributions ensuring that the Open Directory Project will always be free, with public policies, self-regulated by the community and the users as the first priority.

Other examples of collections that might interest us are the free software distributions, with the programs modified so that they fit together perfectly and are precompiled so that they can be run easily.

10.1.6. Hardware

There are two main aspects involved in freedom as regards to hardware. The first one is the need for the interfaces and instruction sets to be free, in such a way that anyone can create a device handler or a compiler for an architecture. The second point is that there should be sufficient information and power available for reproducing a hardware design, modifying it and combining it with others. The designs can be considered software in an appropriate language (VHDL, Verilog, etc). However, making them work is not easy, as they have to be manufactured, which is expensive and slow. However, there are initiatives in this sense, among which we could mention OpenCores (http://www.opencores.org) [52], for integrated circuits.

10.1.7. Literature and art

To finish off our examination of free resources, we cannot forget art and literature, whose ultimate objective is not as much utilitarian as it is aesthetical. What reasons might an artist have to give people the freedom to copy, modify or redistribute their work? On the one hand, it can help to make them well-known and favour the dissemination of their work, which allows them to obtain income from other activities, such as concerts or commissions, and on the other, it can promote experimentation and creativity. In art, we have the same circumstances as in technical subjects. Innovation is incremental and it is sometimes difficult to distinguish between plagiarism and a work that is representative or follows an artistic movement or trend.

Obviously, creation and interpretation are not the same thing, and neither are music and literature. Music, painting, photography and cinema are very similar to programs, in the sense that they can be made to "work" immediately on a computer, whereas the same does not apply to sculpture, for example. There are not many open source initiatives in art and literature and the ones that exist are very diverse. We could mention the novels by the Wu Ming (http://www.wumingfoundation.com) [29] collective.

10.2. Licenses for other free resources

The licenses for free software have been a source of inspiration for other intellectual resources, in such a way that many of them have been adopted directly, especially where documentation is concerned, and on other occasions, they have been adapted slightly, as occurs with the pioneering Open Audio License (http://www.eff.org/IP/Open_licenses/eff_oal.html) [114]. Most of these licenses are *copyleft* licenses, if they permit derived works.

GNU's free documentation license (*see* section 10.2.1) has been used and is often used for all kinds of texts, although the Creative Commons licenses (*see* section 10.2.2) are gradually being accepted.

In fact, program licenses (GPL and LGPL) have even been used for hardware, although this subject is complex and difficult to reconcile with the current law. In effect, the designs and diagrams can be used, without physically being copied, to extract ideas that are used for new closed designs. For example, the OpenIPCore Hardware General Public License ("OpenIPCore hardware general public license") [155] establishes that this appropriation is not permitted, but the legal validity of the document is questionable [209]. The only possible way of protecting these ideas is using some form of free patent, which is something that has not yet developed and is out of the reach of those that do not intend or are unable to establish a business built on the ideas.

10.2.1. GNU free documentation license

One of the most well-known *copyleft* licenses for technical documentation, whether it corresponds to programs or any other matter, is that of the Free Software Foundation. After realising that a document is not the same as a program, Richard Stallman promoted a license for the documents that went with the programs and for other documents of a technical or didactic nature.

In order to smooth the development of the derived versions, a *transparent* copy of the document must be made available to whoever needs it, as explained in section 3.2.5, as well as the *opaque* copies, in an analogy between the source codes and the objects of the programs.

One of the reasons for having a license is to establish authorship and to ensure that the ideas or opinions expressed by the author are not mischaracterised. This is why the derived works must have a title on the cover different to that of the previous versions (unless express permission has been given) and must expressly state the place where the original can be obtained. The names of the main authors of the original documents must also be listed, as well as the names of the people that have made any of the modifications, and all notes on intellectual property must be preserved. Likewise, any acknowledgements and dedications must be preserved and the history section, if there is one, must be respected when new modifications are added. It is even possible, and this is the aspect of the license that has most been criticised, to designate invariant sections and cover texts, which no one can modify or eliminate, although the license only permits *non-technical* texts to be considered as invariant sections, which the license refers to as *secondary* sections.

This license has created a lot of controversy in the free software world, to the point that the Debian distribution project is currently (at the time of publication of this book) discussing whether to eliminate the license or designate all documents that have the license as *non-free* and consider them as non-official. Even though there are no invariant sections, because the derived works may be subject to the terms of the same license, it is important to remember that they could be added subsequently. It is argued, for example, that there may be incorrect or obsolete invariant sections, which, nevertheless, have to be preserved. In any case, the license is incompatible with Debian's free software guidelines (http://www.debian.org/social_contract.html#guidelines) [104], but the question hinges perhaps on whether the documentation must follow these guidelines (for example, the texts of the licenses cannot be modified either).

Advice

The first versions of this text were covered under the GFDL license, but the authors subsequently decided to use too a Creative Commons license (*see* section 10.2.2), which is more appropriate for the characteristics of a book. So this text is a dual licensing work.
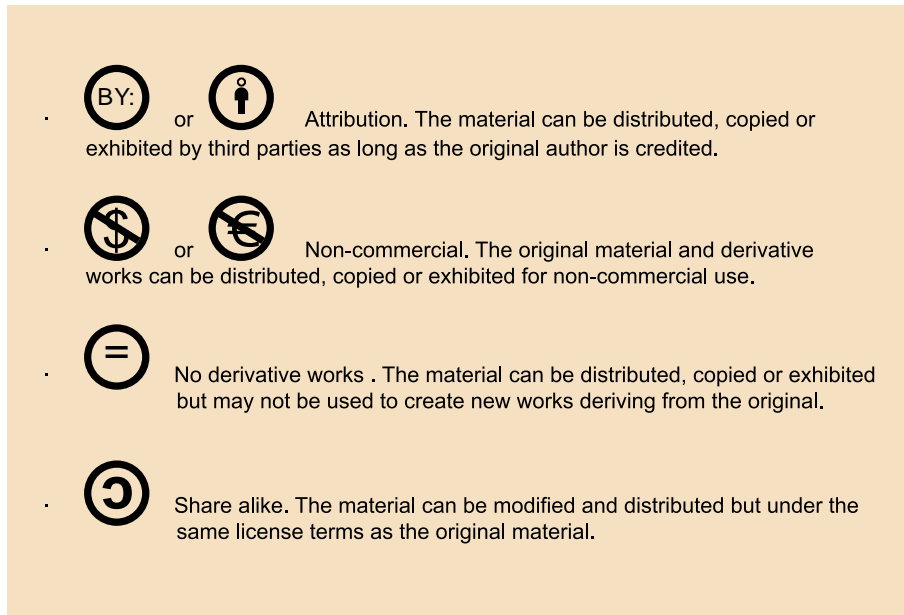
10.2.2. Creative Commons licenses

Creative Commons (http://creativecommons.org) [21] is a non-profit organisation that was founded in 2001 by experts in intellectual property and law in the information society, with the aim of fostering the creation, conservation and accessibility of intellectual resources ceded to the community in numerous ways. It is based on the idea that some people may not wish to make use of all the intellectual property rights that the law grants them, as this could impede their wide distribution.

The first Creative Commons licenses for creative works, of which there were various versions, originally came about in late 2002. These licenses were designed to be:

• strong enough to withstand a court's scrutiny, in many countries;

• simple enough for non-lawyers to use;

• sophisticated enough to be identified by various web applications.

The different licenses allow the creator to select what types of freedoms are allowed, apart from copying, in accordance with four points:

In version 1.x of the Creative Commons licenses, there were eleven types of license, which combined the four basic characteristics mentioned above. 98% of the authors chose the "attribution" option; consequently, as of version 2.x of the Creative Commons licenses, attribution is a requirement. This reduces the eleven types of license to six, which are as follows:

- (BY:) Attribution

- (BY:) (=) Attribution – No derivative works

- (BY:) (=) (S) Attribution – No derivative works – Non-commercial

- (BY:) (S) Attribution – Non-commercial

- (BY:) (S) (↻) Attribution – Non-commercial – Share alike

- (BY:) (↻) Attribution – Share alike
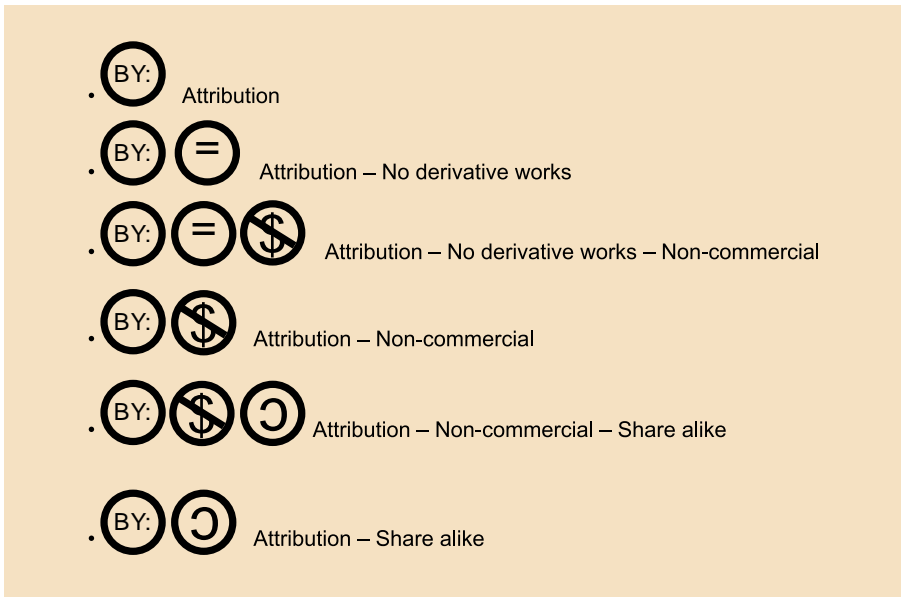
The following table shows a schematic of the licenses with the corresponding icons. This icon is usually a link to a summary of the license, hosted at Creative Commons website[21].

	Allows modifications.	Allows modifications if shared alike.	Does not allow modifications.
Allows commercial use.	CC ① BY	CC ① ↻ BY SA	CC ① = BY ND
Does not allow commercial use.	CC ① ⑤ BY NC	CC ①⑤↻ BY NC SA	CC ①⑤= BY NC ND

It is possible to use the generic icon[14] instead of the icon representing the license, but it must be linked to the license chosen by the author. The HTML code of the link to the license may be obtained from Creative Commons [21]. Once the license has been chosen and the corresponding icon added, the work will have been licensed and you will receive the:

- *Commons deed*. A summary of the license with the relevant icons for it. This summary will be shown when clicking on the link obtained from Creative Commons [21].

- *Legal Code*. This is the complete legal text on which the license is based. This text may be accessed from the summary mentioned above.

- *Digital code*. This is the RDF (*resource description framework*) description, which search engines and other applications can use to identify the license and the terms of use.

In February 2007, version 3.0 of the Creative Commons licenses was published. This is an update that corrects many of the faults that people identified. The first large modification is that the generic license is no longer based on the US model and is now based on the terminology of the Berne Convention. Secondly, moral rights and rights management societies are mentioned specifically, as different rulings had been made in each jurisdiction. Thirdly and finally, the texts of both the *commons deed* and the *legal code* that went with each license were modified to make it clear that the clause on the recognition of authorship does not allow the licensee to imply or give the impression that they have a relationship or are associated in any way with the licensor.

In addition, Creative Commons provides other types of licenses for specific applications. Such as:

- **pd** Public Domain. License used to release the work from copyright completely.

- **dev nations** Developing Nations. The most permissive license for countries considered to be in development by the World Bank.

- **samp-ling** Sampling. License used for sharing snippets (fragments of code which perform a useful function).

- **fc** Founders' Copyright. License used to release the work from copyright after a period of fourteen or twenty-eight years.

- **GNU GPL** CC-GNU GPL. A license which adds the Creative Commons' metadata and summary (Commons Deed) to the Free Software Foundation's GNU General Public License.

- **GNU LGPL** A license which adds the Creative Commons' metadata and summary (Commons Deed) to the Free Software Foundation's GNU General Public License.

- **wiki** Wiki. License for Wiki. In practical terms this is identical to the attribution and share alike license.

- **share music** Music Sharing. License used to share music.

Not all Creative Commons licenses are considered free by sectors linked to free software, as the four essential freedoms must apply before licenses are defined as such (*see* section 1.1.1) Benjamin "Mako" Hill (Debian and Ubuntu developer) created the Freedomdefined.org (http://freedomdefined.org/) [28] website, with the aim of providing a better definition of what is free culture

and what is not. On this basis, of the six basic Creative Commons licenses, only two are strictly free: attribution alone (BY) and attribution-share-alike (BY-SA), the latter of which also has *copyleft*.

Bibliography

[1] Aap Project: http://www.a-a-p.org

[2] Ada Core Technologies: http://www.gnat.com/

[3] Alcôve: http://www.alcove.com

[4] Alcôve-Labs: http://www.alcove-labs.org

[5] Alioth: http://alioth.debian.org

[6] Anjuta: http://www.anjuta.org

[7] The Apache Ant Project: http://ant.apache.org

[8] Arch Revision Control System: http://www.gnu.org/software/gnu-arch/

[9] artofcode LLC: http://artofcode.com/

[10] Autoconf: http://www.gnu.org/software/autoconf

[11] Barrapunto: http://barrapunto.com

[12] Bazaar GPL Distributed Version Control Software: http://bazaar-vcs.org/

[13] Berlios. The Open Source Mediator: http://berlios.de

[14] Bitkeeper Source Management: http://www.bitkeeper.com

[15] Bruce Perens: http://perens.com/OpenStandards/Definition.html

[16] Caldera: http://www.sco.com

[17] Cisco Enterprise Print System: http://ceps.sourceforge.net/

[18] Code::blocks: http://www.codeblocks.org

[19] Commons: http://commons.wikimedia.org/

[20] Concurrent Version System: http://ximbiot.com/cvs/

[21] Creative Commons. http://creativecommons.org

[22] Directory of Open Access Journals: http://www.doaj.org

[23] Eclipse - An Open Development Platform: http://www.eclipse.org

[24] eCos: http://sources.redhat.com/ecos/

[25] eCos license 2.0: http://www.gnu.org/licenses/ecos-license.html

[26] *First Monday. Peer Rewiewed Journal on the Internet*: http://firstmonday.org

[27] Free Software Foundation: http://www.fsf.org

[28] Freedom Defined (Free Cultural Works): http://freedomdefined.org/

[29] Fundación Wu Ming: http://www.wumingfoundation.com

[30] GForge: http://gforge.org

[31] Gettext: http://www.gnu.org/software/gettext

[32] GNU Automake: http://www.gnu.org/software/automake

[33] GNU Emacs: http://www.gnu.org/software/emacs/

[34] GNU Libc: http://www.gnu.org/software/libc

[35] GNU Libtool: http://www.gnu.org/software/libtool

[36] GNU Make: http://www.gnu.org/software/make/make.html

[37] GNU Troff: http://www.gnu.org/software/groff/groff.html

[38] "Have you seen these hackers?": http://www.mozilla.org/MPL/missing.html

[39] "History of TeX": http://www.math.utah.edu/software/plot79/tex/history.html

[40] IBM Public License Version 1.0: http://opensource.org/licesenses/ibmpl.php

[41] Jam Product Information: http://www.perforce.com/jam/jam.html

[42] KDevelop: http://www.kdevelop.org

[43] Launchpad: https://launchpad.net

[44] The Linux Documentation Project: http://www.tldp.org

[45] LinuxCare: http://www.levanta.com

[46] Mailman, the GNU Mailing List Manager: http://www.list.org

[47] The Malone Bug Tracker: https://launchpad.net/products/malone

[48] Metawiki: http://meta.wikimedia.org/

[49] Mozilla Public License 1.1: http://www.mozilla.org/MPL/MPL-1.1.html

[50] Mozilla Tinderbox: http://www.mozilla.org/tinderbox.html

[51] NetBeans: http://www.netbeans.org

[52] Open Cores: http://www.opencores.org

[53] Open Directory Project Social Contract:

[54] Open Source Initiative: http://www.opensource.org

[55] Public Library of Science: http://www.publiclibraryofscience.org

[56] Red Hat: http://www.redhat.com

[57] Savannah: http://savannah.gnu.org and http://savannah.nongnu.org

[58] Slashdot: News for Nerds. http://slashdot.org

[59] Sleepycat License: http://www.sleepycat.com/download/oslicense.html

[60] Sleepycat Software: http://www.sleepycat.com/

[61] SourceForge: Open Source Software Development Website: http://sourceforge.net

[62] Subversion: http://subversion.tigris.org

[63] Texinfo - The GNU Documentation System:

[64] Tigris.org: Open Source Software Engineering: http://tigris.org

[65] W3c Document License:

http://www.w3.org/Consortium/Legal/2002/copyright-documents-20021231

[66] Wiktionary: http://www.wiktionary.org

[67] Wikibooks: http://www.wikibooks.org/

[68] Wikinews: http://wikinews.org/

[69] Wikipedia: http://www.wikipedia.org

[70] Wikiquote: http://www.wikiquote.org

[71] Wikispecies: http://species.wikimedia.org/

[72] Wikiversity: http://wikiversity.org/

[73] X Window System Release 11 License: http://www.x.org/Downloads_terms.html

[74] Ximian: http://www.novell.com/linux/ximian.html

[75] Zope Corporation: http://www.zope.com/

[76] Zope Public License 2.0: http://www.zope.org/Resources/ZPL

[77] Law on Intellectual Property. Spanish Royal Legislative Decree 1/1996, of 12th April (April 1996):

[78] Affero General Public License, 2002: http://www.affero.org/oagpl.html

[79] Law on Intellectual Property. Spanish Law 23/2006, of 7th July (July 2006):

[80] Flossimpact Study. Technical Report, European Comission, 2007: http://flossimpact.eu

[81] ISO JTC 1/SC 34. Standard Generalised Markup Language (SGML, ISO 8879), 1986:

[82] **Antoniades, I.; Samoladas, I.; Stamelos, I.; Bleris, G. L.** "Dynamical simulation models of the open source development process" En: Koch [157].

http://wwwai.wu-wien.ac.at/~koch/oss-book/

[83] **Bailey, E. C.** (1998). *Maximum RPM. Taking the Red Hat package manager to the limit*. http://rikers.org/rpmbook/

[84] **González Barahona, J. M.** (2000). "Software libre, monopolios y otras yerbas". *Todo Linux* (3). http://sinetgy.org/~jgb/articulos/soft-libre-monopolios/

[85] **González Barahona, J. M.** (2002). "¿Qué se hace con mi dinero?". *Todo Linux* (17).

http://sinetgy.org/~jgb/articulos/sobre-administracion/

[86] **González Barahona, J. M.; Robles, G.** Libre Software Engineering Web Site.

http://libresoft.dat.escet.urjc.es/

[87] **González Barahona, J. M.; Robles, G.** (2003, mayo). "Unmounting the *code god* assumption". En: *Proceedings of the Fourth International Conference on eXtreme Programming and Agile Processes in Software Engineering*. Genoa, Italy.

[88] **González Barahona, J. M.; Robles, G.; Ortuño Pérez, M. A.; Rodero Merino, L.; Centeno González, J.; Matellán Olivera, V.; Castro Barbero, E. M.; De las Heras Quirós; P.** "Anatomy of two GNU/Linux distributions". En: Koch [157].

http://wwwai.wu-wien.ac.at/~koch/oss-book/

[89] **Barnson, M. P.** *The Bugzilla guide*.

http://www.bugzilla.org/docs214/html/index.html

[90] **Baudis, P.** "Cogito manual page".

http://www.kernel.org/pub/software/scm/cogito/docs/

[91] **Bezroukov, N.** (1998, diciembre). "A second look at the cathedral and the bazaar". *First Monday*, 4(12).

http://www.firstmonday.org/issues/issue4_12/bezroukov/index.html

[92] **Bodnar, L.** (2003). "Linux distributions. Facts and figures".

http://www.distrowatch.com/stats.php?section=packagemanagement

[93] **Boehm, B. W.** (1981). *Software Engineering Economics*. Prentice Hall.

[94] **Bradner, S.** (1996, october). "The Internet standards process. Revision 3 (rfc 2026, bcp 9)".

http://www.ietf.org/rfc/rfc2026.txt

[95] **Cederqvist, P.; GNU** (1993). "CVS - concurrent versions system". http://www.gnu.org/manual/cvs/index.html

[96] **Collins-Sussman, B.; Fitzpatrick; B. W.; Pilato, C. M.** (2004). *Version control with Subversion.* O'Reilly & Associates (http://www.ora.com).

http://svnbook.red-bean.com/

[97] **Cunningham, W.** "Wiki design principles".

[98] **Dachary, L.** (2001). "Savannah, the next generation".

http://savannah.gnu.org/docs/savannah-plan.html

[99] **Autonomous Government of Andalucía** (2003, March). Decree 72/2003, of 18th March, on Measures to Promote the Knowledge Society in Andalucía.

http://www.andaluciajunta.es/SP/AJ/CDA/Ficheros/ArchivosPdf/DecretoConocimiento.pdf

[100] **De Boor, A.***Pmake. A tutorial*. http://docs.freebsd.org/44doc/psd/12.make/paper.html

[101] **De Icaza, M.** "The story of the GNOME Project".

http://primates.ximian.com/~miguel/gnome-history.html

[102] **Senate of the Republic of France**. Forum sur la proposition de loi tendant à généraliser dans l'administration l'usage d'Internet et de logiciels libres.

http://www.senat.fr/consult/loglibre/index.htm

[103] **De las Heras Quirós, P.; González Barahona, J. M.** (2000). "Iniciativas de las administraciones públicas en relación al software libre". *Bole. TIC, ASTIC magazine* (14).

[104] **Debian.** "Debian free software guidelines".

http://www.debian.org/social_contract.html#guidelines

[105] **Debian.***Debian policy manual*.

http://www.debian.org/doc/debian-policy/

[106] **Debian.** "Debian social contract".

http://www.debian.org/social_contract.html

[107] **Schriftenreihe der KBSt** (2003, July). Leitfaden für die migration von basissoftwarekomponenten auf serverund arbeitsplatzsystemen. Technical report, Koordinierungs-und Beratungsstelle der Bundesregierung für Informationstechnik in der Bundesverwaltung (KBSt).

http://www.kbst.bund.de/download/mlf_v1_de.pdf

[108] **DiBona, C.; Ockman, S.; Stone, M.** (ed.) (1999). *Open sources. Voices from the open source revolution.* O'Reilly & Associates.

http://www.oreilly.com/catalog/opensource/

[109] **Open Directory Project**. http://dmoz.org

[110] **Ehrenkrantz, J. R.** (2003, May). "Release management within open source projects". In: *Proceedings of the 3rd Workshop on Open Source Software Engineering at the 25th International Conference on Software Engineering.*Portland, USA

[111] **European Council** (1991). Council Directive 91/250/CEE of 14th May 1991, on the legal protection of computer programs.

http://europa.eu.int/scadplus/leg/es/lvb/l26027.htm

[112] **Feller, J.; Fitzgerald, B; Hissam, S.; Lakhani, K.** (ed.) (2003). *Making sense of the bazaar*. O'Reilly.

[113] **Fogel, K.; Bar, M.** (2001). *Open source code development with CVS* (2nd edition). Paragliph Press.

http://cvsbook.red-bean.com

[114] **Electronic Frontier Foundation.** Open Audio.

http://www.eff.org/IP/Open_licenses/eff_oal.html

[115] **Free Software Foundation.** GPLv3.

http://gplv3.fsf.org

[116] **Free Software Foundation.** LGPLv3. First discussion draft.

http://gplv3.fsf.org/pipermail/info-gplv3/2006-July/000008.html

[117] **Free Software Foundation** (1985): "The GNU Manifesto".

http://www.gnu.org/philosophy/

[118] **Free Software Foundation** (1991, junio). GNU General Public License, version 2. http://www.fsf.org/licenses/gpl.html

[119] **Free Software Foundation** (1999, February). GNU Lesser General Public License, version 2.1.

http://www.fsf.org/licenses/lgpl.html

[120] **Free Software Foundation.** "Free software definition".

http://www.gnu.org/philosophy/free-sw.html

[121] **Free Software Foundation.** "Free licenses".

http://www.gnu.org/licenses/license-list.html

[122] **Garbee, B.; Koptein, H.; Lohner, N.; Lowe, W.; Mitchell, B.; Murdock, I.; Schulze, M.; Small, C.** "A brief history of Debian". In the package: *Debian-history*.

[123] **Germán, D.** (2002, May). "The evolution of GNOME". In: *Proceedings of the 2nd Workshop on Open Source Software Engineering at the 24th International Conference on Software Engineering*. Florida, USA

[124] **Germán, D.; Mockus, A.** (2003, May): "Automating the measurement of open source projects". In: *Proceedings of the 3rd Workshop on Open Source Software Engineering at the 25th International Conference on Software Engineering*. Portland, USA

[125] **Ghosh, R. A.** (1998, March). "Cooking pot markets: an economic model for the trade in free goods and services on the Internet. *First Monday*, 3(3).

http://www.firstmonday.dk/issues/issue3_3/ghosh/index.html

[126] **Ghosh, R. A.; Glott, R.; Krieger, B.; Robles, G.** (2002). *Free/libre and open source software: Survey and study*. Part iv: "Survey of developers".

http://www.infonomics.nl/FLOSS/report/FLOSS_Final4.pdf

[127] **Ghosh, R. A.; Prakash, V. V.** (2000, July). "The orbiten free software survey". *First Monday*, 5(7).

http://www.firstmonday.dk/issues/issue5_7/ghosh/index.html

[128] **Godfrey, M. W.; Tu, Q.** (2000, August). "Evolution in open source software. A case study". In: *Proceedings of the 2000 International Conference on Software Maintainance*.

[129] **González, J. A.** (2002, March). "Carta al congresista Villanueva".

http://www.gnu.org. pe/mscarta.html

[130] **Goosens, M.; Rahtz, S.** (1999). *The LaTeX Web Companion*. Addison Wesley.

[131] **Grad, B.** (2002, January-March). "A personal recollection: IBM's unbundling of software and services". In: *IEEE Annals of the History of Computing*, 24(1):64-71.

[132] **Working Group on Libre Software** (1999). "Free software / open source. Information society opportunities for Europe?".

http://eu.conecta.it/paper.pdf

[133] **GrULIC.** "Legislation on the use of free software by the State ".

http://proposicion.org.ar/doc/referencias/index.html.es

[134] **Hamerly, J; Paquin, T.; Walton, S.** (1999). "Freeing the source. The story of Mozilla". http://www.oreilly.com/catalog/opensources/book/netrev.html

[135] **Hammel, M. J.** (1991, December). "The history of xfree86". *Linux Magazine*.

http://www.linux-mag.com/2001-12/xfree86_01.html

[136] **Harris, S.** (2001, August). *The Tao of IETF. A novice's guide to the Internet engineering task force* (RFC 3160, FYI 17).

http://www.ietf.org/rfc/rfc3160.txt

[137] **Harrison, P.** (2002). "The rational street performer protocol".

http://www.logarithmic.net/pfh/RSPP

[138] **Hasan, R.** "History of Linux".

http://ragib.hypermart.net/linux/

[139] **Hauben, M.; Hauben, R.** (1997). *Netizens. On the history and impact of Usenet and the Internet*. IEEE Computer Society Press.

[140] **Healy, K.; Schussman; A.** (2003, January). "The ecology of open source software development". http://opensource.mit.edu/papers/healyschussman.pdf

[141] **Hecker, F.** (1998, May). "Setting up shop. The business of open-source software".

http://www.hecker.org/writings/setting-up-shop.html

[142] **Hecker, F.** (1998). "Setting up shop. The business of open-source software".

http://www.hecker.org/writings/setting-up-shop.html

[143] **Hertel, G.; Niedner, S.; Herrmann, S.** (2003). "Motivation of software developers in open source projects. An Internet-based survey of contributors to the Linux kernel".

http://opensource.mit.edu/papers/rp-hertelniednerherrmann.pdf

[144] **Himanen, P.** (2001). *The hacker ethic and the spirit of the information age*. Random House.

http://www.hackerethic.org

[145] **Hunt, F.; Johnson, P.** (2002). "On the Pareto distribution of SourceForge projects. Technical report". Centre for Technology Management, Cambridge University Engineering Department, Mill Lane, Cambridge CB2 1RX.

http://www-mmd.eng.cam.ac.uk/people/fhh10/Sourceforge/Sourceforge%20paper.pdf

[146] **Open Source Initiative.** "History of the OSI".

http://www.opensource.org/docs/history.php

[147] **Hamilton, J. R.** (US ambassador to Peru) (2002, June). "Carta al presidente del Congreso de la República".

http://www.gnu.org.pe/lobbyusa-congreso.html

[148] **Jones, P.** (2000, May). "Brook's law and open source. The more the merrier?".

http://www-106.ibm.com/developerworks/opensource/library/os-merrier.html?dwzone=opensource

[149] **Jorgensen, N.** "Incremental and decentralized integration in FreeBSD". In: Feller *et al.* [112]. http://www.dat.ruc.dk/~nielsj/research/papers/bazaar-freebsd.pdf

[150] **Brooks, F. P.** (1975). *The mythical man-month. Essays on software engineering.* Addison-Wesley.

[151] **Kalt, C.** (2000, April). "Internet relay chat: architecture (RFC 2810)".

http://www.ietf.org/rfc/rfc2810.txt

[152] **Kelsey, J.; Schneier, B.** (1998, November). "The street performer protocol". In: *Third USENIX Workshop on Electronic Commerce Proceedings*. USENIX Press.

http://www.counterpane.com/street_performer.html

[153] **Kelsey, J.; Schneier, B.** (1999, June). "The street performer protocol and digital copyrights". *First Monday*, 4(6).

http://www.firstmonday.dk/issues/issue4_6/kelsey/

[154] **Kelty, C. M.** (2001, December). "Free software/free science". *First Monday*, 6(12).

http://firstmonday.org/issues/issue6_12/kelty/index.html

[155] **Khatib, J.** "OpenIPCore Hardware General Public License".

http://www.opencores.org/OIPC/OHGPL_17.shtml

[156] **Knuth, D.** (1989). *The TeXbook*. Addison Welsley.

[157] **Koch, S.** (ed.) (2003). *Free/open source software development*. Idea Group Inc.

http://wwwai.wu-wien.ac.at/~koch/oss-book/

[158] **Koch, S.; Schneider, G.** (2000). "Results from software engineering research into open source development projects using public data". In: *Diskussionspapiere zum Tätigkeitsfeld Informationsverarbeitung und Informationswirtschaft, H.R. Hansen und W.H. Janko (Hrsg.), Nr. 22*, WirtschaftsuniversitätWien.

[159] **Kovács, G. L.; Drozdik, S.; Succi, G.; Zuliani, P.** (2004). "Open source software for the public administration". In: *Proceedings of the 6th International Workshop on Computer Science and Information Technologies* (CIST 2004). Budapest, Hungary.

[160] **Krishnamurthy, S.** (2002, May). "Cave or community? An empirical examination of 100 mature open source projects". *First Monday*, 7(6).

http://www.firstmonday.dk/issues/issue7_6/krishnamurthy/index.html

[161] **Laffitte; Trégouet; Cabanel** (1999). Proposition de loi numéro 495. Senate of the Republic of France.

http://www.senat.fr/consult/loglibre/texteloi.html

[162] **Laffitte; Trégouet; Cabanel** (2000). Proposition de loi numéro 117. Senate of the Republic of France.

http://www.senat.fr/consult/loglibre/texteloi.html

[163] **Lamport, L.** (1994). *LaTeX user's guide and reference manual* (2nd edition). Addison Welsley, Reading, Mass.

[164] **Lancashire, D.** (2001, December). "Code, culture and cash. The fading altruism of open source development". *First Monday*, 6(12).

http://www.firstmonday.dk/issues/issue6_12/lancashire/index.html

[165] **Lehman, M. M.; Ramil, J. F; Wernick, P. D.** (1997, November). "Metrics and laws of software evolution. The nineties view". In: *Proceedings of the 4th International Symposium on Software Metrics*.

http://www.ece.utexas.edu/~perry/work/papers/feast1.pdf

[166] **Leiner, B. M.; Cerf, V. G.; Kahn, R. E.; Clark, D. D.; Kleinrock, L.; Lynch, D. C.; Postel, J.; Roberts, L. G.; Wolff, S.** (1997). "A brief history of the Internet". In: *Communications of the ACM*.

http://www.isoc.org/internet/history/brief.shtml

[167] **Netcraft Ltd. August 2003 Web Server Survey, 2003.**

http://news.netcraft.com/archives/2003/08/01/august_2003_web_server_survey.html

[168] **Lucovsky, M.** (2000). "From NT OS/2 to Windows 2000 and beyond. A software-engineering odyssey".

http://www.usenix.org/events/usenix-win2000/invitedtalks/lucovsky_html/>

[169] **McGraw, G.** "Building secure software: how to avoid security problems the right way". Cited by: David A. Wheeler in http://www.dwheeler.com/sloc/

[170] **McKusick, M. K.** (1999). "Twenty years of Berkeley Unix. From AT&T owned to freely redistributable". In: DiBona *et al.* [108].

http://www.oreilly.com/catalog/opensources/

[171] **SUN Microsystems** (2000). "Sun microsystems announces availability of StarOffice source code on OpenOffice.org".

http://www.collab.net/news/press/2000/openoffice_live.html

[172] **Mockus, A.; Fielding, R. T.; Herbsleb, J. D.** (2000, June). "A case study of open source software development: the Apache server". In: *Proceedings of the 22nd International Conference on Software Engineering (ICSE 2000)*, pages 263272. Limerick, Ireland ACM Press.

[173] **Molenaar, B.** "What is the context of charityware?".

http://www.moolenaar.net/Charityware.html

[174] **MIT OpenCourseWare.**

http://ocw.mit.edu

[175] **Nagel, L. W.** (1996, september). "The life of SPICE". In: *1996 Bipolar Circuits and Technology Meeting*. Minneapolis, MN, US

http://www.icsl.ucla.edu/aagroup/Life%20of%20SPICE.html

[176] **Narduzzo, A.; Rossi, A.** (2003, May). "Modularity in action: GNU/Linux and free/open source software development model unleashed".

http://opensource.mit.edu/papers/narduzzorossi.pdf

[177] **Newman, N.** (1999). "The origins and future of open source software".

http://www.netaction.org/opensrc/future/

[178] **Nupedia.**

http://www.nupedia.com

[179] **Villanueva Núñez, E.** (2002, April). "Letter to Microsoft Peru".

http://www.gnu.org.pe/rescon.html

[180] **Danish Board of Technology** (2002, October). "Open-source software in e-Government, analysis and recommendations drawn up by a working group under the danish board of technology. Technical report".

[181] **Open Source Initiative.** "Open source licenses".

http://www.opensource.org/licenses/index.html

[182] **Pareto, W.** (1896). "Course of Political Economy". Lausanne.

[183] **Perens, P.; The Open Source Initiative** (1998). "The open source definition". http://www.opensource.org/docs/definition_plain.html

[184] **GNU Peru.** "Proyectos ley de software libre en la Administración pública del Gobierno peruano, Congreso de la República".

http://www.gnu.org.pe/proleyap.html

[185] **Pinheiro, P.** (1999, December). Proposição pl-2269/1999: Dispõe sobre a utilização de programas abertos pelos entes de direito público e de direito privado sob controle acionário da administração pública. Câmara dos Deputados do Brasil.

http://www.camara.gov.br/Internet/sileg/Prop_Detalhe.asp?id=17879

http://www.fenadados.org.br/software.htm

[186] **Pranevich, J.** (2003). "The wonderful world of Linux 2.6".

http://www.kniggit.net/wwol26.html

[187] **The Debian Project.** "Debian developer map".

http://www.debian.org/devel/developers.loc

[188] **Puigcercós Boixassa, J.** (2002). Draft Bill on Measures for Implementing Free Software in Public Administration.

http://www.congreso.es/public_oficiales/L7/CONG/BOCG/B/B_244-01.PDF

[189] **Quittner, J.; Slatalla, M.** (1998). *Speeding the net: the inside story of Netscape and how it challenged Microsoft.* Atlantic Monthly Pr.

[190] **Rasch, C.** "A brief history of free/open source software movement".

http://www.openknowledge.org/writing/open-source/scb/brief-open-source-history.html

[191] **Rasch, C.** (2001, May). "The Wall Street performer protocol. Using software completion bonds to fund open source software development". *First Monday*, 6(6).

[192] **Raymond, E. R.** (2001, January). *The cathedral and the bazaar. Musings on Linux and open source by an accidental revolutionary.* O'Reilly & Associates (http://www.ora.com).

http://catb.org/~esr/writings/cathedral-bazaar/

[193] **Reis, C R.; De Mattos Fortes, R. P.** (2002, February). "An overview of the software engineering process and tools in the Mozilla Project".

http://opensource.mit.edu/papers/reismozilla.pdf

[194] **Rideau, F. R.** (2000). "Patents are an economic absurdity".

http://fare.tunes.org/articles/patents.html

[195] **Roberts, L.** (1978, November). "The evolution of packet switching". *Proceedings of the IEEE*, (66).

[196] **Robles, G.; González Barahona, J. M.; Centeno González, J.; Matellán Olivera, V.; Rodero Merino, L.** (2003, May). "Studying the evolution of libre software projects

using publicly available data". In: *Proceedings of the 3rd Workshop on Open Source Software Engineering at the 25th International Conference on Software Engineering*. Portland, US.

[197] **Robles, G.; Scheider, H.; Tretkowski, I.; Weber, N.** (2001): "Who is doing it? Knowing more about libre software developers".

http://widi.berlios.de/paper/study.pdf

[198] **Rochkind, M.** (1986, May). "Interview with Dick Haight". *Unix Review*.

[199] **Scacchi, W.** (2003). "Understanding open source software evolution. Applying, breaking and rethinking the laws of software evolution".

http://www.ics.uci.edu/~wscacchi/Papers/New/Understanding-OSS-Evolution.pdf

[200] **Schneier, B.** (2000). "Software complexity and security".

http://www.counterpane.com/crypto-gram-0003.html

[201] **Smoogen, S. J.** "The truth behind Red Hat names".

http://www.smoogespace.com/documents/behind_the_names.html

[202] **Haggen So.** "Comparison of free/open source hosting (FOSPhost) sites available for hosting projects externally from project owners".

http://www.ibiblio.org/fosphost/exhost.htm

[203] **Stallman, R.** "GNU coding standards".

http://www.gnu.org/prep/standards.html

[204] **Stallman, R.** "Why *free software* is better than *open source*".

http://www.fsf.org/philosophy/free-software-for-freedom.html

[205] **Stallman, R.** (1998). "Copyleft: pragmatic idealism".

http://www.gnu.org/philosophy/pragmatic.html

[206] **Stallman, R.** (1998). "Why *free software* is better than *open source*".

http://www.gnu.org/philosophy/free-software-for-freedom.html

[207] **Stallman, R.** (1998). "Why software should not have owners".

http://www.gnu.org/philosophy/why-free.html

[208] **Stallman, R.** "The GNU Operating System and the Free Software Movement". In: DiBona *et al.* [108].

http://www.fsf.org/gnu/thegnuproject.html

[209] **Stallman, R.** (1999, June). "On free hardware". *Linux Today*.

http://features.linuxtoday.com/news_story.php3?ltsn=1999-06-22-005-05-NW-LF

[210] **Stallman, R.** (2001). "The free universal encyclopedia and learning resource".

http://www.gnu.org/encyclopedia/free-encyclopedia.html

[211] **Stallman, R.** (2002). *Free software, free society. Selected essays of Richard M. Stallman.* Joshua Gay.

[212] **Stallman, R.** (2003). "Some confusing or loaded words and phrases that are worth avoiding".

http://www.gnu.org/philosophy/words-to-avoid.html

[213] **Stoltz, M.** (1999). "The case for government promotion of open source software".

http://www.netaction.org/opensrc/oss-report.html

[214] **Tanenbaum, A.; Torvalds, L.** (1999). "The Tanenbaum-Torvalds debate".

http://www.oreilly.com/catalog/opensources/book/appa.html

[215] **The Open Source Initiative.** "The open source definition".

http://www.opensource.org/docs/definition_plain.html

[216] **Tiemann, M.** "Future of Cygnus Solutions. An entrepreneur's account". In: DiBona *et al.* [108].

http://www.oreilly.com/catalog/opensources/book/tiemans.html

[217] **Torvalds, L; Diamond; D.** (2001). *Just for fun: the story of an accidental revolutionary.* Texere.

[218] **Linus Torvalds, Hamano, J. C.; Ericsson, A.** "Git manual page".

http://www.kernel.org/pub/software/scm/git/docs/

[219] **Tuomi, I.** (2002). "Evolution of the Linux credits file: methodological challenges and reference data for open source research".

http://www.jrc.es/~tuomiil/articles/EvolutionOfTheLinuxCreditsFile.pdf

[220] **Several authors.** "Open letter to WIPO".

http://www.cptech.org/ip/wipo/kamil-idris-7july2003.pdf

[221] **Vigo i Sallent, P.; Benach i Pascual, E.; Huguet i Biosca; J.** (2002, May). Proposició de llei de programari lliure en el marc de l'Administració pública de Catalunya.

http://www.parlament-cat.es/pdf/06b296.pdf

http://www.hispalinux.es/modules.php?
op=modload&name=Sections&file=index&req=viewarticle&artid=49

[222] **Villanueva Núñez, E.** (2001, December). Free software project bill, number 1609.

http://www.gnu.org.pe/proley1.html

[223] **Villanueva Núñez, E.; Rodrich Ackerman, J.** (2002, April). Bill on the use of free software by the Public Administration, number 2485.

http://www.gnu.org.pe/proley4.html

[224] **W3C** (2000). *Extensible markup language (xml) 1.0* (2nd edition).

[225] **Walsh, N.; Muellner, L.; Stayton, B.** (2002). *DocBook: the definitive guide.* O'Reilly. http://docbook.org/tdg/en/html/docbook.html

[226] **Welke, L; Johnson, L.** (1998). How the ICP Directory began.

http://www.softwarehistory.org/history/Welke1.html

[227] **Wheeler, D. A.** (2000, July). "Estimating Linux's size".

http://www.dwheeler.com/sloc

[228] **Wheeler, D. A.** (2001, June). "More than a gigabuck: estimating GNU/Linux's".

http://www.dwheeler.com/sloc

[229] **Wiesstein, E.** "Concise encyclopedia of mathematics".

http://mathworld.wolfram.com/

[230] **Wikipedia.** "Gini coefficient".

http://www.wikipedia.org/wiki/Gini_coefficient

[231] **Wikipedia.** "Lorenz curve".

http://www.wikipedia.org/wiki/Lorenz_curve

[232] **Wikipedia.** "Pareto".

http://www.wikipedia.org/wiki/Pareto

[233] **Wikipedia.** "TeX".

http://www.wikipedia.org/wiki/TeX

[234] **Wilson, B.** "Netscape Navigator".

http://www.blooberry.com/indexdot/history/netscape.htm

[235] **Computer World** (2000). "Salary survey 2000".

http://www.computerworld.com/cwi/careers/surveysandreports

[236] **Young, R**. (1999). "Giving it away. how Red Hat software stumbled across a new economic model and helped improve an industry".

http://www.oreilly.com/catalog/opensources/book/young.html

[237] **Zawinsky, J. W.** (1999). "Resignation and postmortem".

http://www.jwz.org/gruntle/nomo.html

Appendixes

Jesús M. González Barahona
Joaquín Seoane Pascual
Gregorio Robles

Index

1. Appendix A. Learning guide

A.1. Introduction

What is free software? What is it and what are the implications of a free program licence? How is free software developed? How are free software projects financed and what are the business models associated to them that we are experiencing? What motivates developers, especially volunteers, to become involved in free software projects? What are these developers like? How are their projects coordinated, and what is the software that they produce like? In short, what is the overall panorama of free software?

These are the sort of questions that we will try to answer in this document. Because although free software is increasing its presence in the media and in debates among IT professionals, and although even citizens in general are starting to talk about it, it is still unknown for many people. And even those who are familiar with it are often aware of just some of its features, and mostly ignorant about others.

A.2. Aims

The general aim is, unquestionably, that the reader understand and think logically about basic free software concepts and their main implications. Let us look for more specific aims:

- Knowing what is (and what is not) free software and the main consequences that such a definition has.

- Exploring the rudiments of the legal questions surrounding free software and, particularly, the importance of licenses, the main types and their consequences.

- Having a perspective of the reality of free software, from a global and historical point of view and from the perspective of the most advanced and current projects.

- Learning and getting to know the methods in which free software projects may be financed (when such means exist) and the relevant business models.

- Learning the most important details of the free software development models and the methods for studying them from the perspective of software engineering.

A.3. Contents and learning plans

This text is structured into various chapters (didactic modules) and written in such a way that they are practically independent and self-contained, which means that, excepting the introduction, the book can be read in any order. However, readers are advised to follow the order established for the book, in accordance with the plan below.

The course will be structured in ECTS credits, which means that the planning will require an overall effort form the student, which will include exercises and debates, which will last 150 hours.

Chapter 1 (6 hours). Introductory module discussing all the specific aspects of free software and focusing essentially on an explanation of the underlying basis, for people who are learning about the matter for the first time and on highlighting its importance. An introduction covering the definition of *free software* and its main consequences, amongst other elements, will be provided.

Aims	Content	Materials	Activities	Time
Learning what *freedom* means with regard to software	The four freedoms	Section 1.1.1	Reading the material	2 hours
Distinguishing between *free software* and other related concepts	Definition of related concepts, whether they are similar or analogous	Section 1.1.2	Reading the material and making suggestions	1 hour
Introducing the reasons for which free software is made	Ethical and practical motivations	Section 1.2	Reading the material and making suggestions	1 hour
Introducing the consequences of free software	Consequences for the user, the State, the developer, etc.	Section 1.3	Reading the material and making suggestions	2 hours

Chapter 2 (14 hours). Historical development of the world of free software, from its beginning in the seventies to the current moment, offering a broad vision of the most notable milestones, the main projects, the financial, professional or social evolution, etc.

Aims	Content	Materials	Activities	Time
Learning about the "prehistory" of free software	Facts before the existence of the concept	Section 2.1 and beginning of annex B	Reading the material and making suggestions	2 hours
Learning about the history of free software all the way up to the present day	Most significant events in chronological order	Sections 2.2, 2.3, 2.4 and rest of annex B	Reading the material and making suggestions	10 hours
Trying to predict the future	Some predictions (hopes and problems)	Section 2.5	Reading the material and making suggestions	2 hours

Chapter 3 (9 hours). Legal aspects of free software. The most common free
software licenses and their effects on business and development models will
be analysed in detail.

Aims	Content	Materials	Activities	Time
Learning the basic concepts of intellectual and industrial property	Copyright, intellectual property, patents, brands, industrial secrets	Section 3.1	Reading the material and making suggestions	3 hours
Learning the legal basis of free software: the licenses	Definition of free licenses and the features of the most important licenses	Section 3.2	Reading the material and making suggestions	7 hours

Chapter 4 (8 hours). Characteristics of free software developers and the motivations that lead them to participate in the projects, thereby making the existence of free programs possible.

Aims	Content	Materials	Activities	Time
Getting to know the type of people that develop free software	Ages, genders, professions, geographical location, etc.	Sections 4.1, 4.2, 4.3 and 4.4	Reading the material and making suggestions	4 hours
Learning how much time to spend on it and why	Weekly dedication, motivations, questions of prestige and leadership	Sections 4.5, 4.6, 4.7 and 4.8	Reading the material and making suggestions	4 hours

Chapter 5 (22 hours). Financial aspects of free software and, especially, methods for financing the projects and business models that are being explored.

Aims	Content	Materials	Activities	Time
Learning about the sources of finance	Financial sources used	Section 5.1	Reading the material and making suggestions	8 hours
Learning how to profit from free software	Business models	Sections 5.2 and 5.3	Reading the material and making suggestions	8 hours
Learning about the relationship between free software and the monopolistic situations that are typical in the software industry	Monopolies and software. Free software's role	Sections 5.1, 5.2, 5.3 and 5.4	Reading the material and making suggestions	6 hours

Chapter 6 (28 hours). Relationship of policies and free software and, especially,
policies for promoting free software and the use of free software by public
administrations.

Aims	Content	Materials	Activities	Time
Learning about the effect of free software on public administrations.	Main effects and difficulties in implementation	Section 6.1	Reading the material and making suggestions	4 hours
Learning about what administrations do or can do with regard to free software	Solutions to needs, promotion and investment in R&D	Section 6.2	Reading the material and making suggestions	4 hours

Aims	Content	Mate-rials	Activities	Time
Learning about legislative initiatives	Revision of legislative initiatives for implementing or supporting free software, including examples of specific texts.	Section 6.3	Reading the material and making suggestions	20 hours

Chapter 7 (12 hours). Management and development models for free software projects, techniques that have been successful and quantitative and qualitative studies of free software from the perspective of development.

Aims	Content	Materials	Activities	Time
Learning about the paradigmatic models of software development	"The cathedral and the bazaar"	Sections 7.1, 7.2, 7.3 and 7.5	Reading the material and suggested bibliography	3 hours
Learning about the processes involved in the development of free software	Characteristic processes	Section 7.4	Reading the material and suggested bibliography	3 hours
Learning about the possibilities and realities that the availability of sources and the associated registries bring to the free software engineering	Resources and quantitative studies	Section 7.6	Reading the material and suggested bibliography	3 hours
Learning what remains to be done in free software engineering	Future tasks	Section 7.7	Reading the material and suggested bibliography	3 hours

Chapter 8 (14 hours). Introduction of the technologies and development environments for free software and their effects on the management and evolution of the projects.

Aims	Content	Materials	Activities	Time
Learning the general features of the environments and the tools that free software developers use	General characterisation	Section 8.1	Reading the material and suggested bibliography	1/2 hour
Learning the basic development tools	Languages, compilers, operating systems, etc.	Section 8.2 and 8.3	Reading the material and suggested bibliography	2 hours
Learning the basic methods with which developers work together	Messaging, forums, repositories, chats and *wikis*	Section 8.4	Reading the material and suggested bibliography	2 hours
Learning the mechanisms used to manage sources and their versions	CVS and new alternatives	Section 8.5	Reading the material and suggested bibliography	4 hours
Learning how free software is documented	Languages and tools for documentation	Section 8.6	Reading the material and suggested bibliography	2 hours
Learning how errors and tasks are managed	Bug management systems	Section 8.7	Reading the material and suggested bibliography	1 hour
Learning how portability is supported	Resources for other architectures	Section 8.8	Reading the material and suggested bibliography	1/2 hour
Learning about the public environments of integrated development	SourceForge and others	Section 8.9	Reading the material and suggested bibliography	2 hours

Chapter 9 (30 hours). Studying free software projects (revising the most interesting classical free software projects, in terms of results obtained, management model, historical evolution, effect on other projects, etc.). Study of companies related to free software.

Aims	Content	Materials	Activities	Time
Learning an example of operating systems	Linux and *BSD	Sections 9.1 and 9.2	Reading the material and suggested bibliography	8 hours
Learning an example of desktop environments	Gnome and KDE	Sections 9.3 and 9.4	Reading the material and suggested bibliography	8 hours
Learning an example of system programs	Apache	Section 9.5	Reading the material and suggested bibliography	2 hours
Learning an example of end user programs	Mozilla and OpenOffice	Sections 9.6 and 9.7	Reading the material and suggested bibliography	4 hours
Learning an example of a distribution	Red Hat and Debian	Sections 9.8 and 9.9	Reading the material and suggested bibliography	8 hours

Chapter 10 (6 hours). Module in which free resources other than software are presented; these are resources that have been created partly thanks to free software and the model that it has given.

Aims	Content	Materials	Activities	Time
Learning other free resources	Free texts, hardware, teaching materials and art	Section 10.1	Reading the material and suggested bibliography	3 hours
Learning about the applicable licenses	Licenses, especially the Creative Commons licenses	Section 10.2	Reading the material and suggested bibliography	3 hours

2. Appendix B. Key dates in the history of free software

This is only a list of the dates that could be considered to be important in the history of free software. It is based on the one that appears in [132] and the one provided by the Open Source Initiative [146] and is not supposed to be comprehensive: there are certainly many important dates that have not been included in the list. However, we hope to provide a sufficiently complete view of the historical landscape in which the world of free software has evolved.

Dates	Events
1950s and 1960s	The software is distributed with its source code and without any restrictions on the user groups such as SHARE (IBM) and DECUS (DEC).
1969, April	RFC number 1, which describes the first Internet (then called ARPANET) is published. The free availability of the RFCs and, particularly, of the specifications of the protocols used in Internet were key factors for its development.
1970, January	IBM began selling its software separately, creating the beginning of the proprietary software industry.
1972	Unix begins to be distributed in universities and research centres.
1973	Unix arrives at Berkeley University, in California. The history of Unix BSD begins.
1973	SPICE is placed by Donald O. Penderson in the public domain. With time, it will become the standard in its field (integrated circuit simulators).
1978	Donald Knuth, of Stanford University, starts working on TeX, an electronic typesetting system that will be distributed as free software.
1983	Richard Stallman writes "The GNU Manifesto", in which he asks for software to be shared with the public again.
1984	The GNU project begins. The developers that work on it, initially coordinated by Richard Stallman, begin to create a large number of tools similar to those in Unix, including an editor (Emacs) and a compiler (GCC). The aim is to build an operating system that is completely free.
1985	The X Consortium, based at MIT, distributes the X Window system as free software, under a license that is hardly restrictive at all.
1985	Richard Stallman founds the Free Software Foundation. Among other tasks, the Foundation will work as a centre that receives the funds and resources that will assist the development of the GNU project and as the owner of the intellectual property generated by the project.
1989	Cygnus, the first company that essentially provides commercial services for free software (including support, development and adaptation of free programs), is founded.
1989	The Network Simulator (or simply, ns) begins to be developed as a variant of the REAL Network Simulator. Ns is a free telecommunication network simulator that will be used extensively by universities all over the world and that will become a standard in its field, to a certain extent.
1990	The Free Software Foundation announces that it intends to build a kernel that will be called GNU Hurd. The aim of this project is to complete what the GNU project's strategy was most missing: a complete operating system.
1991	William and Lynne Jolitz write a series in *Dr. Dobbs Journal* on how to port BSD Unix to PC based on the i386.

Dates	Events
1991, August	Linus Torvalds, a twenty-one year old Finnish student announces that he has begun work on a free Unix-type kernel using GNU tools, such as GCC. His aim at the time is to build a *free Minix*.
1991, October	Linus Torvalds releases the first version of his kernel, which is still very primitive and is called Linux.
1992	The US Air Force awards New York University a contract to build an open source compiler for the new version of Ada (a language that it was almost obligatory to use at that time in all contracts with the US military), Ada 95. The NYU team chooses GNU GCC for the generation of code and calls its compiler GNAT (GNU NYU Ada 95 Translator).
1992, July	William and Lynne Jolitz release 386BSD 0.1, which, with time, will give rise to the projects NetBSD, FreeBSD and later OpenBSD.
1993	SuSE is founded in Germany, which begins its business distributing Slackware Linux, translated into German.
1993, August	Ian Murdock starts a new distribution based on Linux called Debian GNU/Linux, which will become the distribution built by voluntary developers with the most participants.
1993, December	FreeBSD 1.0, one of the first stable distributions derived from the Jolitz's 386BSD is released on the Internet.
1994	The GNAT developers found the company Ada Core Technologies, with the aim of guaranteeing its development and evolution in the future and with a business model based on providing services to their clients' compiler (and not selling the compiler itself, which continues to be free software). With time, GNAT will become the leader in the market of Ada compilers.
1994, January	Version 0.91 of Debian GNU/Linux is released; it is the fruit of the efforts of twelve developers.
1994, March	The first edition of the *Linux Journal* is published.
1994, 29th July	Marc Ewing publishes the first version of Red Hat Linux. As is the case with Debian, the aim is to improve the results of the predominant distribution in that time, Slackware.
1994, October	NetBSD 1.0. is released
1995	Bob Young founds Red Hat Software buying the Red Hat Linux distribution from its creator, Marc Ewing, and merging it with his own business, ACC, which has been selling materials related to Linux and Unix through catalogue since 1993. A little later, Red Hat Linux 2.0 is released; it is the first distribution that includes the RPM packaging format.
1995	DARPA supports the development of ns through the VINT project.
1995, January	FreeBSD 2.0. is released
1995, April	The first official release of Apache (0.6.2) takes place.
1996	The First Conference on Freely Redistributable Software takes place in Cambridge, Massachusetts, US.
1996, October	The KDE project is announced; it is one of the first to address usability problems in the Unix environment and the first that tries to do so on a large scale in the world of free software.
1997, January	Eric S. Raymond presents his paper "The cathedral and the bazaar", in which he expresses his opinions on why certain free software development models work.
1997, August	Miguel de Icaza announces the GNOME project, a *competitor* to KDE with similar aims, but with the explicit objective of ensuring that the whole of the resulting system is free software. Born as a reaction of the Free Software Foundation and others to the licensing problems that KDE had, which involved a fundamental component, the Qt library, which was not free software at that time.
1998, 22nd January	Netscape declares its intention of distributing as free software the code of its browser (Netscape Navigator), which had been the leader in the web browser market.

Dates	Events
1998, 3rd February	Chris Peterson, Todd Anderson, John Hall, Larry Augustin, Sam Ockman and Eric Raymond meet up to study the consequences of Netscape's announcement with regard to the release of its browser and decide to promote the term *open source software* [146], using it as a brand that guarantees that the products that have it consist of free software. The promoters of this term understand that it is more appropriate for the corporate world than the one that was more commonly used up to that moment, *free software*. The Open Source Initiative is created to manage the term.
1998, 31st March	Netscape publishes a large part of its source code for Netscape Navigator on the Internet.
1998, 7th May	Corel announces the NetWinder, a *network computer* based on Linux. It is the first time that a large company commercialises an element that uses software that is basically free software. Shortly afterwards, Corel announces its plan to port its office software (which includes WordPerfect) to Linux, which is also a novelty for the time.
1998, 28th May	Sun Microsystems and Adaptec become part of Linux International. They are the first big IT companies to do so.
1998, June	The technical conference of USENIX, which is usually dedicated to Unix, opens a parallel session called FREENIX, focusing on free software.
1998, 22nd June	IBM announces that it will commercialise and provide support for Apache, using it as the server of its WebSphere product line.
1998, July	Debian GNU/Linux 2.0 is released; it has been built by more than three hundred volunteers and the distribution includes more than one thousand five hundred packages.
1998, July	KDE 1.0 is released; it is the first version distributed as *stable*. Several GNU/Linux distributions incorporate it shortly afterwards.
1998, August	Linus Torvalds and Linux appear on the cover of *Forbes* magazine.
1998, 29th September	Red Hat, which is the leading company in the market of Linux-based distributions at the time, announces that Intel and Netscape have bought a minority share in its capital. Free software begins to awaken interest among investors.
1998, November	MandrakeSoft is founded and shortly afterwards, it releases Mandrake Linux, its distribution of GNU/Linux.
1998, 1st November	The *Halloween Documents*, in which Microsoft supposedly identifies GNU/Linux and free software as an important competitor and plans how to attack it, are published.
1999, 27th January	HP and SGI announce that they will support Linux in their computers, which marks the beginning of a trend: the abandonment of proprietary Unix by the computer manufacturers that used them as *their* operating system, in favour of Linux.
1999, March	GNOME 1.0, which will subsequently be made more stable (October GNOME) and incorporated in several GNU/Linux distributions, is released.
1999, 9th March	Debian GNU/Linux 2.1 is released, with more than two thousand packages.
1999, 15th March	Apple releases Darwin, which will be the central component of its new Mac OS X, under a free license.
1999, August	Red Hat is floated on the stock exchange. The price of the shares increases enormously in the first days after the float, to the extent that it is capitalised at 4,800 million dollars. Later, other companies related to free software, such as VA Linux and Andover.net, will also be floated on the stock exchange. The value of the shares of all these companies will plummet a few years later, when the dotcom bubble explodes; many of these companies will not survive the event.
1999, October	Two companies are founded in order to produce software in the framework of the GNOME project: Eazel (which will go bankrupt in 2002, after producing Nautilus, a file manager) and Helix Code (later renamed Ximian and subsequently bought by Novell, which will produce tools such as Red Carpet or Evolution).
1999, November	Red Hat Software buys Cygnus. The resulting company is the biggest company in the world in the field of free software.

Dates	Events
2000, January	Mozilla M13, considered by many as the first reasonably stable version of Mozilla, is released almost two years after the release of a large part of Netscape Navigator's code.
2000, May	GNOME 1.2 (Bongo GNOME) is released.
2000, August	The creation of the GNOME Foundation is announced.
2000, 15th August	Debian GNU/Linux 2.2 is released, with more than two thousand five hundred source packages, which comprise approximately 55 million lines of code.
2001, January	Version 2.4 of Linux is released.
2001, 15th January	Wikipedia is started. The idea of building an encyclopaedia using a *wiki* as IT support, where, in principle, anyone can cooperate, applying working methods that are very similar to those used in free software, becomes a reality.
2002, 30th January	ObjectWeb, an organisation founded in France by Bull, France Telecom and INRIA that is one of the first organisations designed to produce free software by cooperating with companies and research centres, is founded with clearly commercial objectives and the idea of being the nucleus of an international community of interests.
2002, 3rd April	KDE 3.0, the third generation of the KDE desktop environment, is released. The quality of free desktops begins to match that of traditional commercial desktops.
2002, April	The gnuLinEx project is publicly announced; with this project, the Regional Government of Extremadura (Spain) wishes to use its own GNU/Linux distribution in the computers of all the public schools in the region.
2002, May	Mozilla 1.0, the first officially stable version of the project, is released.
2002, 1st May	The office suite, OpenOffice.org 1.0, is released; it will soon become a standard office application suite in the free software world.
2002, 26th June	GNOME 2.0, which represents an important step forward for users, with a more carefully designed interface and more attention to user-friendliness, is released. Other aspects that improve the accessibility are also introduced.
2002, 19th July	Debian GNU/Linux 3.0 is released with more than 100 million lines of source code; more than nine hundred developers participate in this version.
2002, 28th July	Version 3.0 of Knoppix is released; it is an evaluation distribution that can be installed on a hard disk quickly and easily, and it becomes a tremendous success.
2002, 23rd September	The first version of Firefox (which is called Phoenix at the time) is released, as an experimental extension based on the code of Mozilla Suite that is supposed to be simpler.
2002, December	Red Hat Softwre announces that its cash flow in the second and third quarters of 2002 was positive.
2002, 16th December	The first Creative Commons licenses are published (although the project was launched in 2001).
2003, January	MandrakeSoft, a company that produces the Mandrake Linux distribution, declares bankruptcy.
2003, 19th January	FreeBSD 5.0-RELEASE is released, after almost three years of work since the previous stable large-scale version.
2003, 22nd January	The number of articles in English on Wikipedia reaches one hundred thousand articles. Shortly afterwards, the number of German articles reaches ten thousand.
2003, February	Motorola begins selling the A760 in China; it is the first mobile telephone that uses an operating system based on Linux (the MontaVista Linux distribution).
2003, 6th March	The SCO group files a lawsuit against IBM for *devaluing* its version of Unix. This marks the beginning of a lawsuit in which IBM is accused of contributing code that belongs to SCO to the Linux kernel.

Dates	Events
2003, 28th May	Munich City Council (Germany) announces that Linux will replace Windows in most of its computer systems.
2003, July	MandrakeSoft announces that its finances have been positive for the whole year and that it expects to come out of receivership in late 2003.
2003, 7th July	An open letter [220] is written to the WIPO (World Intellectual Property Organization) asking it to examine new open models of collaborative creation (including free software but also the Human Genome project or open scientific journals).
2003, 15th July	The Mozilla Foundation is established. Netscape Inc. (now the property of AOL) announces that it will no longer develop the Netscape browser and, therefore, it will no longer work on the Mozilla project. The Mozilla Foundation is established with a donation of two million dollars from AOL and material support and human resources from various companies, including AOL itself, Red Hat and Sun Microsystems.
2003, 4th August	Novell buys Ximian Inc., one of the leading companies in the development of free software (especially for GNOME), as part of its strategy to establish itself in the market for Linux-related solutions.
2003, 2nd September	OpenOffice.org 1.1 is released.
2003, 24th September	The European Parliament amends the Directive on Patentability of Computer-Implemented Inventions so that (if it is approved as it stands) software patents are not allowed in the European Union. The Directive, which was originally proposed by the European Commission precisely to ensure that these types of patents were legal, is still in the codecision procedure, in which the Council of Ministers will also have to provide its opinion.
2003, 5th November	Version 1 (FC1) of de Fedora Core, the fruit of the communal development process that Red Hat had announced a few months before, is released. As of this moment, the company Red Hat will commercialise Red Hat Enterprise Linux, whilst the Fedora Core collections are not officially maintained by Red Hat, but by the community of voluntary developers that build it with the assistance of Red Hat (which already existed before Red Hat decides on this collaboration).
2004, 13th January	Novell finishes its purchase of SuSE for a total of 210 million dollars.
2004, 9th February	The Mozilla Foundation decides to change the Mozilla Firebird name (previously called Phoenix) to Mozilla Firefox. This will be the definitive name of the browser, while its development is close to version 1.0.
2004, 18th May	The European Council, as part of the codecision process on the European Directive on the Patentability of Computer-Implemented Inventions, decides to submit a *compromise* version of the text to the European Parliament; however, it is accused of ignoring the Parliament's vote, as the new version permits the patenting of software. The decision is so contentious, even within the Council itself, that it is not formally approved until March 2005.
2004, 8th September	Pepper Computer announces that it will launch the first miniPC with a touch screen that uses an operating system that is completely free, based on Fedora Core.
2004, 20th September	The number of articles on Wikipedia reaches one million, in one hundred and five languages.
2004, 20th October	The first version of Ubuntu is released; it is based on Debian and the aim is to publish new versions regularly. The construction of the distribution is financed by the company Canonical, which offers maintenance and services for the distribution. The distribution will become very successful, fairly quickly.
2004, 9th November	Version 1.0 of Firefox is released, after a long series of preparatory versions. This version was downloaded more than 25 million times in the one hundred days following its release.
2005, 24th January	MandrakeSoft announces that it is buying the Brazilian company Conectiva, which releases a distribution based on linux with the same name. Shortly afterwards, MandrakeSoft announces that it is changing its name to Mandriva.
2005, 1st May	OASIS recognises ODF (*open document format*), the data format use by OpenOffice.org 2.0, among others, as a standard.

Dates	Events
2005, 25th May	Nokia announces its Nokia 770, a miniPC that uses a version of Debian GNU/Linux with the X Window system and GTK+.
2005, 6th June	Debian GNU/Linux 3.1 is released; it now has more than 200 million lines of source code.
2005, 14th June	Sun Microsystems releases Open Solaris, the free version of its Solaris operating system.
2005, 15th June	Mandriva buys the US company Lycoris (previously called Redmond Linux) and begins working on a distribution that incorporates the previous versions of Mandrake, Conectiva and Lycoris.
2005, 6th ly	The European Parliament rejects the proposal of the Directive on the Patentability of Computer-Implemented Inventions received from the Council of Ministers, during the second reading. This means that the only legal text applicable to the subject in the European Union is the European Patent Convention of 1973.
2005, 20th October	Version 2.0 of OpenOffice.org, which is distributed under the LGPL, is released.
2005, December	The first version of Ruby on Rails, a work environment for the development of web applications using the model-view-controller architecture, is released. Distributed with license X11, it will be widely used in the prototyping and development of numerous web services.
2005, December	Nicholas Negroponte announces the OLPC (One Laptop Per Child) project, which has the aim of designing and building a portable PC of 100 dollars for children in developing countries. It uses free software with a GNU/Linux version called Sugar, based on Red Hat.
2005, 14th December	The science journal *Nature* publishes a paper comparing Wikipedia with the *Encyclopaedia Britannica*; according to the paper, the degree of precision with regard to scientific subjects of both encyclopaedias is similar.
2006, 16th January	The first draft of the GPLv3 is published; it is an attempt to update the GPL, which is the license that is most commonly used for free software projects at the time (and by a long way). At this point, an open debating process begins with regard to the changes.
2006, 1st March	The number of articles in English on Wikipedia reaches one million.
2006, 20th March	Fedora Core 5 is released.
2006, 1st June	Ubuntu 6.06 LTS is released; it is advertised as being supported by the company Canonical for three years.
2006, August	The number of Firefox downloads reaches 200 million (there are many more downloads from unofficial sites, which are not taken into account). Around this time, it is estimated that the browser has a 12% share of the global market (approximately 20% in Europe).
2006, 12th November	Sun announces that it will release the different versions of the Java platform under the GPL. Up until this moment, these versions had been distributed for free in binary, which Sun had justified citing compatibility and stability issues; however this has made it extremely difficult to use Java in free software distributions.
2006, 30th November	The ISO (International Standards Organization) and the IEC (International Electrotechnical Commission) jointly publish OASIS' ODF version as an international standard (ISO/IEC 26300:2006) for the exchange of office information.
2006, December	The Taiwanese company First International Computer (FIC) presents the first advanced mobile telephone based on code that is completely open, in the Open Source in Mobile conference. It is called Neo1973, it costs 350 dollars and it uses a software platform called OpenMoko, based on the kernel of Linux 2.6, GTK+, X Windows and Matchbox.
2007, January	The FLOSSImpact [80] study, on the effect (especially the economic effect) of free software, is published. The study has been financed by the European Commission and it is the first large-scale study in the field.
2007, 23rd February	Version 3.0 of the Creative Commons Licenses is published.

Dates	Events
2007, 8th pril	Version 4.0 of Debian GNU/Linux is released.

3. Appendix C. GNU Public License

Version 2, June 1991

Copyright© 1989, 1991 Free Software Foundation, Inc. 675 Mass Ave, Cambridge, MA 02139, USA

Literal copies of this document may be copied and distributed, but not modified.

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 Lesser 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:

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.

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.

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.

2) 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,

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 condi-

tions. 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 ver-

sion 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 an idea of what it does. Copyright (C) *yyyy 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., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, 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) *yearname 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 Lesser General Public License instead of this License.

4. Appendix D. Texts of some legislative proposals and related documents

Below is the literal text of some of the legislative proposals mentioned in chapter 6 and of some of the related documents.

D.1. Draft bill brought by Laffitte, Trégouët and Cabanel (France)

We provide below a translation of the proposed law made in October 1999 by the French senators Pierre Laffitte, René Trégouët and Guy Cabanel [laffitte99:_propos].

D.1.1. Recitals

(Only the paragraphs on free software are included.)

[...] In order to guarantee the perpetuity of accessible data, facilitate its exchange and ensure that citizens have free access to information, the use of this information by the Administration must not depend on the goodwill of software manufacturers. It is necessary to have free systems whose development may be guaranteed thanks to the manufacturers' source code being available to all.

Free software is currently developing very fast. There are many IT companies that recognise that the future of their business is not in selling software, but in assisting people that use it, by providing the associated services.

Our bill would establish that, after a transitional period defined by decree, the use of free software will be obligatory in all public administrations.

Proprietary software, whose source code is not freely available, may only be used in specific cases, when an authorisation is provided by a free software agency. [...]

D.1.2. Articles

- Article 1. On the dematerialisation of information and data exchange between public administrations.
  State services, local administrations and public bodies will ensure that their information and data are put into an electronic format, with electronic networks, as of 1st January 2002.

The conditions that regulate the transition from the current paper-based exchange and the future exchange using electronic formats and networks will be specified by decree.

- Article 2. On the dematerialisation of public market processes.
  In order to guarantee a great degree of transparency and quick access to information for companies, all public tenders and the attached documents, will be published in electronic formats and networks, as of 1st January 2002. Likewise, all bids for public tenders must be published in electronic formats and networks.
  A decree will determine the mechanisms of the transition to electronic processes.

- Article 3. On open technologies.
  Subject to the exceptions mentioned in article 4, as of 1st January 2002, State services, local administrations and public bodies may only use software that is free to use and modify and for which the source code is available.
  A decree will determine the terms and conditions of the transition.

- Article 4. On the Free Software Agency.
  A Free Software Agency will be created. It will be in charge of informing the State services, local administration and public bodies of the conditions in which this law must be applied. The Agency will determine the use of software licenses that are appropriate in the context established by this law.
  The Agency will ensure the interoperability of the free software used by the public administrations.
  The Agency will make an inventory, for each sector, of any fields in which there is no available free software, no applicable software that can be freely used and modified or no applicable software whose source code is available. On the basis of this inventory, the Agency will declare the relevant public administrations as exempt from this law.
  The Free Software Agency will be open to all Internet users, and their decisions must be preceded by consultations made on the Internet.
  A representative of the Free Software Agency will be appointed in each prefecture.
  The Free Software Agency's methods of working will be established by decree.

- Article 5. On the dissemination of the modifications to the software used in the context of this law.
  The Free Software Agency will ensure, whilst respecting copyrights, that the modifications to the software are disseminated in accordance with the framework of this law.

- Article 6.

  The costs incurred by the State as a result of this law will be compensated through increases in the rights defined in articles 575 and 575A of the General Tax Code.

D.2. Draft Bill of Le Déaut, Paul and Cohen (France)

We will now provide a translation of practically the whole of the draft bill presented by Jean-Yves Le Déaut, Christian Paul and Pierre Cohen in April 2000.

D.2.1. Recitals

The tremendous growth in the use of new information technologies and telecommunications has made it necessary to produce accompanying legislation. The public services and the local administrations must become the model and engine of the information society that will guarantee individual freedoms, consumer safety and equal opportunities in the field in question.

Various examples show that, despite some significant progress achieved thanks to the actions of the Government in the field of the information society, the State services tend to use communication standards that are intimately linked to one single private provider, which means that a user or collective is bound to act as the client of this same provider, thereby strongly reinforcing the phenomena of abuse of dominant position.

The State service often use software with source code that is not available, which makes it impossible to correct the bugs and faults that the suppliers themselves refuse to correct or check whether there are security deficiencies in sensitive applications. The State services use, sometimes unknowingly, software that secretly transmits information that is *a priori* considered confidential, to foreign societies or organisations.

However, the economic models of the software and telecommunications industry developed by the market are based, to a large extent, on the appropriation of clientele and the exponential valuation of the obtainment of user profiles. These economic models reward strategies of providing incompatible products, of industrial secrets and of planned obsolescence and the violation of individual freedoms. Although the French State cannot eliminate these underlying tendencies using the law due to the transnational nature of communication networks, it can, however, facilitate the development of an information society on French soil that is respectful of public freedoms, of consumer safety and of equal opportunities, and this would hopefully set a precedent for Europe and the world.

The law is based on five principles: a citizen's right to have free access to public information, the perpetuity of public data, the security of the State, consumer safety in the information society and the principle of software interoperability.

In order to guarantee the citizen's free access to public information, the code of the computerised data provided by the Administration must not be linked to one single supplier. Open standards, in other words, those in which the data coding regulations are public, make it possible to guarantee free access, as they permit, where necessary, the development of free compatible software.

In order to guarantee the perpetuity of the public data, the use and maintenance of the software must not depend on the goodwill of the software's creators. It is necessary to have systems whose development is always guaranteed by the availability of the source code. The principle of source code availability in the framework of license-based contracts, which is a principle that to date has only been present as an option in the legislation on public utility and software package purchases, must become the rule and be applied to all public software purchases.

We have deliberately avoided an ambiguous legislative approach based exclusively on the use of free software. It would not be appropriate for the State, regardless of the recognised quality of the free software, to favour a determined economic model for the publication of software. On the contrary, the obligatory resort to open communication standards and the publication of source code will guarantee equal opportunities, in accordance with the principles of interoperability of the legislation on software.

In order to guarantee national security, it is necessary to have systems that are free of elements that may provide remote control of the system or the involuntary transmission of information to any third parties. We need systems whose source code is freely accessible to the public, so that it can be examined by a large number of independent world experts. The bill that we propose should provide more security for the State, as full working knowledge of the source code would eliminate the growing number of pieces of software containing "backdoors".

The bill that we propose would likewise reinforce consumer safety in the information society, as it would allow for the emergence of new offers of software without "backdoors", which would not threaten the right to a private life and individual freedoms.

But for equal opportunities to emerge, it will be necessary to reaffirm and reinforce the principle of interoperability in the legislation on software and legislation on compatibility. Today, both of these rights are threatened by the parties that benefit from their dominant monopolistic position, who put obstacles to avoid the emergence of any competition.

In order to guarantee the interoperability of software, the intellectual or industrial property rights of a software creator must not block the development of new compatible software that would compete with him. The right to compatibility for all, in other words, the right to freely develop, publish and use original software that is compatible with other software, must be guaranteed by the law. Likewise, the principle of interoperability introduced by European laws on software must prevail over the other intellectual or industrial property rights that may apply. Particularly, the existence of a brand on a communications standard or a patent on an industrial process that is necessary to implement a communications standard, must not permit its owner to block or limit the free dissemination of compatible free software.

The bill that we propose could be applied immediately. In effect, most software editors are prepared to adopt open communication standards, such as those defined in Paris, Boston and Tokyo by the World Wide Web Consortium. There are many proprietary software editors that are likewise prepared to provide the French Government with the source code of their products. In addition, the offer of free software based on the Linux operating system will cover many of the Administration's needs, now and in the future. However, the Administrations and its collective bodies are not sufficiently informed about the existence of open standards or the offers of software published with its source code.

In order to facilitate the fast implementation of free standards, it is necessary to reinforce the role of the Inter-ministerial Commission on Technical Support for the Development of Information Technologies and Communication in the Administration (Mission Interministérielle de Soutin Technique pour le Développement des Technologies de l'Information et de la Communication dans l'Administration), and entrust it with the mission of carrying out and disseminating within the Administration, a census of the offer of open standards and software published with its source code. If there is no market for this, the MTIC will be in charge of developing new standards or new software published with its source code. In order to carry out these new tasks, the MTIC will be transformed into the Agency of Information Technologies and Communication (AITC).

When there is no market, the AITC will be in charge of developing new standards or new software published with its source code. In order to ensure equal opportunities, the software developments that occur will be put in the public domain; therefore, these developments may be sold as proprietary software or as free software, according to the license freely chosen by the editor. The AITC will also be in charge of evaluating the levels of interoperability, perpetuity and security of the software purchased by the French Administration.

More generally, the open communication systems and the availability of the source code are essential to guarantee the interoperability, on a European level, between the IT systems of the different administrations and the nation-

al public bodies, and to avoid that the interconnection between systems depend solely on the goodwill of the software editors. The AITC will also be in charge of participating in the international cooperation projects in the sphere of information technologies and communications, and of facilitating interoperability with the information systems of the other European Union member countries.

The bill that we propose would cover the concerns listed above. It reminds us that the State can play an important role in the economy, preserving national and European interests, whilst defending the market economy. This bill would allow France to stand as the defender of freedom within the new information and communication technologies.

D.2.2. Articles

- Article 1.
  For all computerised data exchanges, the State Administration, the local administrations and the local bodies would have the obligation of using open communication standards, constituted by public regulations and procedures for exchanging digital data.

- Article 2.
  The Administration, the public bodies and the territorial public administrations are obliged to use software whose source code is accessible.

- Article 3.
  All individuals or corporate entities have the right to develop, publish or use original software that is compatible with the communication standards of any other software.

- Article 4.
  A public State body will be created, called the Agency of Information Technologies and Communications. This body would report to the Ministry of Industry. The AITC will have the task of reporting to and advising the State services, the collective bodies and the public bodies on the creation and identification of the technical requirements with regard to information and communication technologies. It will identify the needs of the public services with regard to equipment and software, ensure that the communication standards are harmonised and propose the technical practices that must be applied. It will carry out inventories in each sector of activity of the open standards and the available software.
  Depending on the results of the inventory, it will support the development of open standards and software published with its source code and promote the use of this type of software in the public domain to mitigate any deficiencies in the market.
  The AITC will favour the interoperability with the information systems of other EU member States and participate in the international cooperation

projects in the sphere of information and communication technologies.

The AITC will have a representative in each prefecture.

The AITC's ways of working will be established by decree.

- Article 5.

  The modes of applying this law, as well as the conditions of the transition from the current situation, will be established by decree issued by the Council of State.

- Article 6.

  The expenses incurred by the State as a result of applying this law will be paid using the sums established in articles 575 and 575A of the General Tax Code.

D.3. Bill proposed by Villanueva and Rodrich (Peru)

We will now provide the translation into English of the literal text of most of Draft Bill number 2485, on the Law on Free Software in Public Agencies, of the Peruvian congressmen Edgar Villanueva Núñez and Jacques Rodrich Ackerman [223].

D.3.1. Recitals

The complexity of the world we are living in demands permanent review and constant adaptation of its institutional framework to be up to date with the current technological trends that the world imposes.

The discovery of new information technologies and among them, free software, has become an ideal instrument to assure the preservation of the State's data.

In this way technology fulfils its role of facilitating the different and multiple human activities, being one of them, the handling of public information.

According to the Peruvian Constitution, in section 5 of article 2, "all persons have the right to solicit information that they need without disclosing the reason, and to receive that information from any public entity within the period specified by law, at a reasonable cost. Information that affects personal intimacy and that is expressly excluded by law or for reasons of national security is not subject to disclosure".

Section 6 of the same article emphasises the right all persons have "to be assured that information services, computerised or not, public or private, do not provide information that affects personal and family privacy".

Having said this, it is obvious the concern of our Constitution for establishing institutional bases that protect the citizens' freedom to information access and the non-disclosure of information that affects personal and familiar intimacy, likewise for reasons of national security.

The guarantee of these rights in our Constitution isn't solely based in the goodwill of the State's agents to fulfil the norms of the Constitution, but also by the use of technologies that in some cases contribute and in others do not, to an effective protection of said citizens' rights.

It is in this context that it is of utmost importance for the State the incorporation of those technologies that help to reinforce the exercise of the citizens' access to information and its due reserve in cases that require so.

The use of free software in all of the State's agencies points in this direction. Basically we can say that the fundamental principles that drive the present Bill are tightly related to the basic guarantees of a democratic State and we can sum them up in the following:

1) Free access of the citizens to public information

2) Perpetuity of public data

3) Security of the State and of the citizens

To guarantee the citizens' free access to public information, it is essential that the coding of the data is not tied to a sole provider. The use of standard and open formats assures this free access, making possible the creation of compatible software.

To guarantee the perpetuity of public data, it is indispensable that the use and maintenance of software do not depend on the goodwill of the providers, nor on monopoly conditions, imposed by those. Systems whose evolution can be guaranteed by the availableness of source code are needed.

To guarantee national security it's vital to have systems that are devoid of elements that allow remote control or the transmission of non-desired information to third-parties. Therefore, it is essential to have systems whose source code is freely accessible to the public, so that its inspection is allowed to the State, the citizens and a great number of independent experts in the world.

This proposal provides more security, because the knowledge of the source code will eliminate the growing number of programs with spy-ware.

In the same way, this Bill furthers the security of the citizens, both in their condition of legitimate holders of the information handled by the State as in their condition of consumers. In this last case it would allow the growth of an extensive supply of free software devoid of potential spy-ware that makes it possible to jeopardise private life and individual freedoms.

The State, looking to improve the quality of public administration as both keeper and manager of private information, will establish the conditions in which agencies of the State will acquire software in the future, that is, in a manner that is compatible with the constitutional guarantees and basic principles previously stated.

The project clearly states that any given software in order to be acceptable for the State must not only be technically adequate to carry out a given task, but must also fulfil some requirements in license matters, without which the State could not guarantee the citizens the adequate process of their data, looking over for their integrity, confidentiality and permanent accessibility, all of which are critical elements for its fulfilment.

The State establishes conditions for the use of software by the agencies of the State, without meddling in any way in the transactions of the private sector. It is acknowledged that the State does not have the ample spectrum of contractual freedom that the private sector has, because it is restricted due to the requirement of transparency of all public acts, and in this sense the common benefit must be the leading factor to take into account when legislating over this matter.

The project also guarantees the principle of equality before the Law, because no natural or legal entity is excluded of the right to purvey those goods, under the conditions stated in this Bill and without any more limitations than the ones that are established in the Bill of Contracts and Acquisitions of the State (TUO Supreme Decree number 012-2001-PCM).

Additionally to these advantages we could highlight benefits that would begin to show up as a consequence of these measures, immediately after being carried out.

To begin with, there are the job opportunities for local programmers. Of the universe of server Software commercialised in the USA. over the last year, 27% belongs to "free" software, a truly significant portion for that huge and competitive market. The number speaks for itself and constitutes a firm answer to those who would think that free software would imply a hefty limitation to the employment of programmers of the country. On the contrary, the initiative will allow the release of a great amount of resources, and an incentive to boost human creativity.

By making use of free software, professionals can analyse the root of the problems and improve the development in whatever cases are necessary, using the globally available free software, under different licenses. It is an ideal area to employ creativity, an aspect in which young Peruvians would be able to reach good levels.

On the other hand, by means of the free software we get rid of illegal software that is present in some agencies of the State. The non-permitted use of software inside the State or the mere suspicion of this constitutes a powerful incentive to make any given public employee modify the situation that goes against intellectual property.

Although it is correct to say that the adoption of free software is not necessary to abide by the law, its use will drastically reduce the irregular occurrences and will act as a *medium of legal infection*, both in the State and the private sector.

We can count many countries that are formally acknowledging an exclusive use of Free Software in the public sector.

Among them we have France, where a legal norm about this subject is being debated. The government of the city of Mexico (DF) has already started an important migration to adopt free software in a general way and this is the leading country in this field in the western world. Also, in Brazil, the State of Recife has ruled its adoption. The Popular Republic of China has been using free software for several years as a policy of the State. The same applies to Scandinavian countries. In the US both NASA and the US Navy among other organisations have adopted free software for some of their needs, as have also done so other government and private entities.

Finally, the project grants the execution of this law to the Presidency of the Council of Ministers for being this organism the one that concentrates the direction of all government institutions. In this sense it has a strategic advantage for carrying out the given reform and the migratory process of proprietary software to free software.

These are the types of ideas in which these aspects have been specified in this legislative proposal.

D.3.2. Cost/Benefit analysis

This initiative does not imply any expense to the national treasury. However, for the fulfilment of its aims, it will be necessary to reassign the governmental expenditure whose incidence confines itself to what is effectively expended by each governmental organism in the processes of contracts and bids of the State for the acquisition of software.

Although it is true that free software represents a substantial saving for the State's economy, when compared with proprietary software, this is not the central point of support of this Bill. As we have pointed out, its advantage focuses on the technological reassurances that the program conveys to the information that the State handles, information that in many instances is of a reserved nature.

In this sense a better protection of the citizens' rights constitutes a non-measurable benefit that must be taking into account from the cost/benefit analysis point of view.

We can sum up the benefits of the project in the following subjects:

- National Security.
  In order to perform its functions, the State must store and process information on its citizens. The relationship between the individual and the State depends on the privacy and integrity of this data, which must be adequately kept against three specific risks:
  1) Disclosure risk: confidential data must be handled in such way that the access to them is made possible only to authorised persons and institutions.
  2) Risk of impossibility of access: the data must be stored in such way that the access to them by authorised persons and institutions is guaranteed for all its period of usefulness.
  3) Risk of alteration: the alteration of data must be restricted, again only to those authorised to do so.
  With free software all these risks are considerably mitigated. It allows the user to make a complete and exhaustive inspection of the mechanisms that are used to process data. The fact that free software allows the inspection of its sources is an excellent security measure because having the mechanisms exposed to the eyes of trained professionals makes hiding malicious functions inside them exponentially more difficult, even if the end user does not take the time to search for them by himself.

- Technological Independence.
  With proprietary software there is no freedom of contract in the aspects of extension and correction of the system in use, a technological dependence is forged, one in which the provider is in the position of ruling, one-way only, terms, deadlines and costs.
  Free Software entitles the users with the freedom to control, correct and modify the program to suit it better to their needs. This freedom is not aimed at programmers only. Although they are the ones who can take advantage of it first-hand, the users benefit greatly too, for in this way they can hire any programmer (not necessarily the original author) to correct given errors or add functionality.

- Local development.

In the case of proprietary software, the user is able to execute or run a program, but not to inspect or modify it; consequently, the user cannot learn from it; the users become dependent on a technology that not only do they not understand but that is expressly prohibited to them. The professionals in their environment, who could help the users to achieve their aims, are equally limited: as the way in which the program works is secret and its inspection is not permitted, it is not possible to fix it. In this way, local professionals see their possibilities of offering added value constantly more limited and their employment horizons narrow, along with their chances to learn more. With free software, these disadvantages of proprietary software are enormously mitigated.

- Cost of software.
  The cost is greatly reduced because, being free, there is no need to ask for additional licenses to continue using the program. This need does exist with proprietary software. It is important for the user to be able to keep these costs under control, because if he cannot, he might be impeded to further carry on with his goals, bound by unplanned occurrences. Again, here it is the technological dependence that threatens free software.

- More sources of employment.
  With free software, handwork that was chained as a consequence of the technological dependency of the State to proprietary software is freed. Now user resources (in this case the State agencies) will be assigned for maintenance and support of free software.

- Boost to creativity and entrepreneurship.

D.3.2.1. Costs

The big cost that is involved with the change from proprietary to free software is limited to the migratory process. Even if it is true that the migratory process involves costs in studies, decision making to implement the new systems, handwork to implement the change, data conversion, retraining of personal and eventually expenses in licenses and/or development and time; it is no less certain that all these are fixed costs and are paid only once.

On the other hand, proprietary software has its costs, which were paid and which cannot be recovered. But aside from these costs there are others involved with proprietary software: permanent updates (sometimes reinforced by a self-supported monopoly) and above all the huge price for the State that is the loss of the freedoms that guarantee the control of its own information. These costs are permanent and with the passage of time, sooner or later they exceed the fixed costs of carrying out a migration.

To summarise, the benefits of the migratory process exceed its costs.

D.3.3. Legal Formula

D.3.3.1. Article 1. Aim of the law

Employ exclusively free software in all the systems and computer equipment of every State agency.

D.3.3.2. Article 2. Scope of application

The Executive, Legislative and Judicial branches as well as the autonomous regional or local decentralised organisms and the corporations where the State holds the majority of the shares will use free software in their systems and computer equipment.

D.3.3.3. Article 3. Authority of application

The authority in charge to execute the law shall be the Council of Ministers.

D.3.3.4. Article 4. Definition of *free software*

For the purposes of this law, *program* or *free software* shall be defined as that whose license shall guarantee the user, without additional cost, the following:

- Unrestricted use of the program for any purpose.

- Unrestricted access to the respective source code.

- Exhaustive inspection of the working mechanisms of the program.

- Use of the internal mechanisms and arbitrary portions of the software, to adapt them to the needs of the user.

- Freedom to make and distribute copies of the software.

- Modification of the software and freedom to distribute said modifications of the new resulting software, under the same license of the original software.

D.3.3.5. Article 5. Exceptions

Given the case where no solution which uses free software exists, that could satisfy the determined necessity, the State Agencies could adopt the following alternatives adhering to their order.

If verifiable time constraints should occur in attending a technical problem and proprietary software was found to be available, the organism that needed it could negotiate a permission of exception before the competent authority to utilise proprietary software that has the following characteristics:

- The programs shall comply with the stipulations mentioned in section 4 of the law, except for the free distribution of the modified program. In such a case the permission of exception could be definitive.

- If no programs of the preceding category were available, those that exist in a free project of advanced development shall be chosen. The permission in this case shall be transitory and will automatically expire when the free software becomes mature with the functionality that is necessary.

- If no products could be found that met these conditions, then proprietary software could be used, but the demanded permission of exception from the competent authority will expire automatically two years after it was issued, having to be renewed previous establishment that a satisfactory solution of free software was not available.

The competent authority shall emit a permission of exception only if the State organism guarantees the storage of data in open formats, without prejudice of payment for the proprietary licenses.

D.3.3.6. Article 6. Educational permissions

All educational establishments that depend on the State are able to manage its proprietary software license of its use in research, after paying the corresponding intellectual property rights and applicable licenses, provided that the aim of the research is directly associated to the use of the program in question.

D.3.3.7. Article 7. Transparency of the exceptions

The exceptions that originate in the authority of a given application must be sustained and published in the website of the State's Portal.

The resolution that authorises the exception must enumerate the functional requirements that the program must fulfil.

D.3.3.8. Article 8. Exceptional authorisation

In case some State agency cannot fulfil its requirements with software stated in article 2 of this law then it is authorised to acquire proprietary software to store or process data which must be kept in reserve, the respective authority must publish in the State's portal a report where the risks associated with the use of given software for a particular application must be explained.

The exceptional permissions granted to State agencies related with security and national defence are exempted from the previously stated obligation.

D.3.3.9. Article 9. Responsibilities

The maximum administrative authority and the technical and informative authority of each agencies of the State assume the responsibility for the fulfilment of this law.

D.3.3.10. Article 10. Regulatory norm

The executive branch of the government will rule within one hundred and eighty days deadline, the conditions, deadlines and forms in which the current status quo will be changed to one which satisfies the conditions of this law, and will guide, in that sense, all future contracts and negotiations for software acquisition.

In the same way, it will direct the migratory process of the proprietary software systems to free software ones, in every case where the given circumstances so demand.

D.3.3.11. Article 11. Glossary of terms

a) Program or *software*: any sequence of instructions used by a digital data processing system to carry out a specific task or to solve a given problem.

b) Execution or use of a program: the act of using it on any digital data processing system to carry out a function.

c) User: natural or legal entity that makes use of the software.

d) Source code or source program: complete set of instructions and source digital files created or modified by those who programmed them, plus all the support digital files, like data tables, images, specifications, documentation, and any other element that is necessary to create the executable program. As an exception, all those tools that are usually available as free software in other media may be excluded, for example, compilers, operating systems and libraries.

e) Free software or program: that which guarantees the user, without further cost, the following:

- Unrestricted use of the program for any purpose.

- Unrestricted access to the respective source code.

- Exhaustive inspection of the working mechanisms of the program.

- Use of the internal mechanisms and arbitrary portions of the software, to adapt them to the needs of the user.

- Freedom to make and distribute copies of the software.

- Modification of the software and freedom to distribute said modifications of the new resulting software, under the same license of the original software.

f) Proprietary software (non-free software), that which does not fulfil all the requirements listed in the previous statement.

g) Open format: any manner of digitally coded information that satisfies both existent standards and the following conditions:

- Its technical documentation is publicly available.

- The source code of at least one complete reference implementation is publicly available.

- There are no restrictions for the creation of programs that store, transmit, receive or access data codified in such way.

D.4. Letters from Microsoft Peru and congressman Villanueva

On 21st March 2002, Juan Alberto González, the general manager of Microsoft Peru, sent a letter to congressman Edgar Villanueva Núñez with regard to his draft bill on free software [129]. On 8th April, the congressman replied [179]. We include here the English translation of a literal transcription of almost the whole text of both letters (the paragraphs not related to the draft bill have been edited out).

D.4.1. Letter from Microsoft Peru

As we arranged in our meeting, we attended the forum organised in the Congress of the Republic on March 6th regarding the law that you have proposed. There we got the chance to listen to several presentations on the subject. We would now like to present our position so that you have a better view of the real situation.

Your proposal mandates that every public organisation exclusively uses free software, also known as open source software. This is something which transgresses the principles of equality before the law, of no discrimination, of free private initiative, and of freedom of industry and contracting, which are protected by the Constitution.

Your proposal, by making mandatory the use of open source software, establishes discriminatory and non-competitive treatment in contracting and acquisitions by public organisations, violating the basic principles of the Law of State Contracting and Acquisitions (Number 26850).

By forcing the State to favour a business model supporting exclusively open source software, your proposal will discourage local and international software manufacturers who make real and important investments in the country, create a significant number of direct and indirect jobs, and thus contribute to the national income. In contrast, open source software development always has a lesser benefit to the economy, since it mainly creates jobs in the service sector.

Your proposal imposes the use of open source software without considering the risks this carries to security, warranty, and possible violation of the intellectual property rights of third parties.

It erroneously assumes that *open source software* is free software, that is, without cost, and therefore arrives at incorrect conclusions about money saved by the State. It has no cost-benefit analysis to back up this assumption.

It is wrong to think that open source software is free. Research by the Gartner Group (an important market researcher in the technology world, well-known worldwide) has shown that the cost of software acquisition (operating system and applications) is only 8% of the total cost of ownership that enterprises and organisations must face as a consequence of the rational and productive use of technology. The other 92% is costs of installation, training, support, maintenance, management, and downtime.

One of the arguments supporting your proposal is the supposed cheapness of open source software when compared to commercial software, without considering the possibility of volume licensing models. The State can really benefit from these, as other countries have.

Additionally, the approach chosen by your project (i) is clearly more expensive because of the high costs of migration; (ii) risks loss of interoperability among information systems, both inside the State and between the State and the private sector, due to the many different distributions of open source software on the market.

In most cases, open source software does not offer adequate levels of service to achieve better productivity by its users, nor does it offer warranties from well-known manufacturers. These things have caused many public entities to go back on their decisions to use open source software; they are now using commercial software in its place.

This project discourages creativity in the Peruvian software industry, which sells USD 40 million worth of goods every year, USD 4 million of that exported (10th place in the ranking of Peruvian non-traditional exports, more than handcrafted goods) and is a source of highly skilled jobs. With a law encouraging the use of open source software, programmers lose their intellectual property rights and their most important source of remuneration.

Since open source software can be freely distributed, it cannot make any money for its developers by exportation. In this way, it weakens the multiplier effect of software sales to other countries and stunts the growth of this local industry, which the State should be stimulating.

In the forum, the importance of the use of open source software in education was discussed, without commenting on the complete failure of this initiative in countries like Mexico. There, the same State officials who supported the project now say that open source software did not provide a learning experience to children in schools, adequate levels of training were not available nationwide, inadequate support for the platform was provided, and the software was not integrated well enough with existing school computer systems.

If open source software fulfils all the requirements of State entities, why should a law be needed to adopt its use? Should not the market freely choose which products provide more benefits and value?

D.4.2. Reply from Congressman Villanueva

First of all, I thank you for your letter of March 25, 2002 in which you state the official position of Microsoft relative to Bill Number 1609, Free Software in Public Administration, which is indubitably inspired by the desire for Peru to find a suitable place in the global technological context. In the same spirit, and convinced that we will find the best solutions through an exchange of clear and open ideas, I will take this opportunity to reply to the commentaries included in your letter.

While acknowledging that opinions such as yours constitute a significant contribution, it would have been even more worthwhile for me if, rather than formulating objections of a general nature (which we will analyse in detail later) you had gathered solid arguments for the advantages that proprietary

software could bring to the Peruvian State, and to its citizens in general, since this would have allowed a more enlightening exchange in respect of each of our positions.

With the aim of creating an orderly debate, we will assume that what you call *open source software* is what the bill defines as *free software*, since there exists software for which the source code is distributed together with the program, but which does not fall within the definition established by the bill; and that what you call *commercial software* is what the bill defines as *proprietary* or *non-free*, given that there exists free software which is sold in the market for a price like any other good or service.

It is also necessary to make it clear that the aim of the bill we are discussing is not directly related to the amount of direct savings that can be made by using free software in state institutions. That is in any case a marginal aggregate value, but in no way is it the chief focus of the bill. The basic principles which inspire the bill are linked to the basic guarantees of a state of law, such as:

• free access to public information by the citizen,

• permanence of public data,

• security of the State and citizens.

To guarantee the citizens' free access to public information, it is essential that the coding of the data is not tied to a sole provider. The use of standard and open formats gives a guarantee of this free access, if necessary through the creation of compatible free software.

To guarantee the permanence of public data, it is necessary that the usability and maintenance of the software does not depend on the goodwill of the suppliers, or on the monopoly conditions imposed by them. For this reason the State needs systems the development of which can be guaranteed due to the availability of the source code.

To guarantee national security or the security of the State, it is indispensable to be able to rely on systems without elements which allow control from a distance or the undesired transmission of information to third parties. Therefore, systems with source code freely accessible to the public are required to allow their inspection by the State itself, by the citizens, and by a large number of independent experts throughout the world. Our proposal brings further security, since the knowledge of the source code will eliminate the growing number of programs with *spy code*.

In the same way, our proposal strengthens the security of the citizens, both in their role as legitimate owners of information managed by the State, and in their role as consumers; in this second case, by allowing the growth of a widespread availability of free software not containing *spy code* able to put at risk privacy and individual freedoms.

In this sense, the bill is limited to establishing the conditions under which the State bodies will obtain software in the future, that is, in a way compatible with these basic principles.

From reading the bill it will be clear that once passed:

• the law does not forbid the production of proprietary software

• the law does not forbid the sale of proprietary software

• the law does not specify which concrete software to use

• the law does not dictate the supplier from whom software will be bought

• the law does not limit the terms under which a software product can be licensed.

What the bill does express clearly, is that, for software to be acceptable for the State it is not enough that it is technically capable of fulfilling a task, but that further the contractual conditions must satisfy a series of requirements regarding the license, without which the State cannot guarantee the citizen adequate processing of his data, watching over its integrity, confidentiality, and accessibility throughout time, as these are very critical aspects for its normal functioning.

We agree, Mr. González, that information and communication technology have a significant impact on the quality of life of the citizens (whether it be positive or negative). We surely also agree that the basic values I have pointed out above are fundamental in a democratic state like Peru. So we are very interested to know of any other way of guaranteeing these principles, other than through the use of free software in the terms defined by the bill.

As for the observations you have made, we will now go on to analyse them in detail:

Firstly, you point out that: 1. "Your proposal mandates that every public organisation exclusively use free software, also known as open source software. This is something which transgresses the principles of equality before the law, of no discrimination, of free private initiative, and of freedom of industry and contracting, which are protected by the Constitution."

This understanding is in error. The bill in no way affects the rights you list; it limits itself entirely to establishing conditions for the use of software on the part of state institutions, without in any way meddling in private sector transactions. It is a well established principle that the State does not enjoy the wide spectrum of contractual freedom of the private sector, as it is limited in its actions precisely by the requirement for transparency of public acts; and in this sense, the preservation of the greater common interest must prevail when legislating on the matter.

The bill protects equality under the law, since no natural or legal entity is excluded from the right of offering these goods to the State under the conditions defined in the bill and without more limitations than those established by the Law of State Contracts and Purchasing (TUO by Supreme Decree No. 012-2001-PCM).

The bill does not introduce any discrimination whatsoever, since it only establishes *how* the goods have to be provided (which is a State power) and not *who* has to provide them (which would effectively be discriminatory, if restrictions based on national origin, race religion, ideology, sexual preference etc. were imposed). On the contrary, the bill is decidedly anti-discriminatory. This is so because by defining with no room for doubt the conditions for the provision of software, it prevents State bodies from using software which has a license including discriminatory conditions.

It should be obvious from the preceding two paragraphs that the bill does not harm free private enterprise, since the latter can always choose under what conditions it will produce software; some of these will be acceptable to the State, and others will not be since they contradict the guarantee of the basic principles listed above. This free initiative is of course compatible with the freedom of industry and freedom of contract (in the limited form in which the State can exercise the latter). Any private subject can produce software under the conditions which the State requires, or can refrain from doing so. Nobody is forced to adopt a model of production, but if they wish to provide software to the State, they must provide the mechanisms which guarantee the basic principles, and which are those described in the bill.

By way of an example: nothing in the text of the bill would prevent your company offering the State bodies an office *suite*, under the conditions defined in the bill and setting the price that you consider satisfactory. If you did not, it would not be due to restrictions imposed by the law, but to business decisions relative to the method of commercialising your products, decisions with which the State is not involved.

To continue, you note that: 2. "Your proposal, by making mandatory the use of open source software, establishes discriminatory and non-competitive treatment in contracting and acquisitions by public organizations...".

This statement is just a reiteration of the previous one, and so the response can be found above. However, let us concern ourselves for a moment with your comment regarding "non-competitive practices."

Of course, in defining any kind of purchase, the buyer sets conditions which relate to the proposed use of the good or service. From the start, this excludes certain manufacturers from the possibility of competing, but does not exclude them *a priori*, but rather based on a series of principles determined by the autonomous will of the purchaser, and so the process takes place in conformance with the law. And in the bill it is established that *no-one* is excluded from competing as far as he guarantees the fulfilment of the basic principles.

Furthermore, the bill *stimulates* competition, since it tends to generate a supply of software with better conditions of usability, and to better existing work, in a model of continuous improvement.

On the other hand, the central aspect of competition is the chance to provide better choices to the consumer. Now, it is impossible to ignore the fact that marketing does not play a neutral role when the product is offered on the market (since accepting the opposite would lead one to suppose that firms' expenses in marketing lack any sense), and that therefore a significant expense under this heading can influence the decisions of the purchaser. This influence of marketing is in large measure reduced by the bill that we are backing, since the choice within the framework proposed is based on the *technical merits* of the product and not on the effort put into commercialisation by the producer; in this sense, competition is increased, since the smallest software producer can compete on equal terms with the most powerful corporations.

It is necessary to stress that there is no position more anti-competitive than that of the big software producers, which frequently abuse their dominant position, since in innumerable cases they propose as a solution to problems raised by users: "update your software to the new version" (at the user's expense, naturally); furthermore, it is common to find arbitrary cessation of technical help for products, which, in the provider's judgement alone, are *old*; and so, to receive any kind of technical assistance, the user finds himself forced to migrate to new versions (with non-trivial costs, especially as changes in hardware platform are often involved). And as the whole infrastructure is based on proprietary data formats, the user stays *trapped* in the need to continue using products from the same supplier, or to make the huge effort to change to another environment (probably also proprietary).

You add: 3. "By forcing the State to favour a business model supporting exclusively open source software, your proposal will discourage local and international software manufacturers who make real and important investments in the country, create a significant number of direct and indirect jobs, and thus

contribute to the national income. *In contrast*, open source software development always has a lesser benefit to the economy, since it mainly creates jobs in the service sector."

I do not agree with your statement. Partly because of what you yourself point out in paragraph 6 of your letter, regarding the relative weight of services in the context of software use. This contradiction alone would invalidate your position. The service model, adopted by a large number of companies in the software industry, is much larger in economic terms, and with a tendency to increase, than the licensing of programs.

On the other hand, the private sector of the economy has the widest possible freedom to choose the economic model which best suits its interests, even if this freedom of choice is often obscured subliminally by the disproportionate expenditure on marketing by the producers of proprietary software.

In addition, a reading of your opinion would lead to the conclusion that the State market is crucial and essential for the proprietary software industry, to such a point that the choice made by the State in this bill would completely eliminate the market for these firms. If that is true, we can deduce that the State must be subsidising the proprietary software industry. In the unlikely event that this were true, the State would have the right to apply the subsidies in the area it considers of greatest social value; it is undeniable, in this improbable hypothesis, that if the State decided to subsidise software, it would have to do so choosing the free over the proprietary, considering its social effect and the rational use of taxpayer's money.

In respect of the jobs generated by proprietary software in countries like ours, these mainly concern technical tasks of little aggregate value; at the local level, the technicians who provide support for proprietary software produced by transnational companies do not have the possibility of fixing *bugs*, not necessarily for lack of technical capability or of talent, but because they do not have access to the source code to fix it. With free software one creates more technically qualified employment and a framework of free competence where success is only tied to the ability to offer good technical support and quality of service, one stimulates the market, and one increases the shared fund of knowledge, opening up alternatives to generate services of greater total value and a higher quality level, to the benefit of all involved: producers, service organisations, and consumers.

It is a common phenomenon in developing countries that local software industries obtain the majority of their takings in the service sector, or in the creation of *ad hoc* software. Therefore, any negative impact that the application of the bill might have in this sector will be more than compensated by a growth in demand for services (as long as these are carried out to high quality standards). If the transnational software companies decide not to compete under these new rules of the game, it is likely that they will undergo some

decrease in takings in terms of payment for licences; however, considering that these firms continue to allege that much of the software used by the State has been illegally copied, one can see that the impact will not be very serious. Certainly, in any case their fortune will be determined by market laws, changes in which cannot be avoided; many firms traditionally associated with proprietary software have already set out on the road (supported by copious expense) of providing services associated with free software, which shows that the models are not mutually exclusive.

With this bill the State is deciding that it needs to preserve certain fundamental values. And it is deciding this based on its sovereign power, without affecting any of the constitutional guarantees. If these values could be guaranteed without having to choose a particular economic model, the effects of the law would be even more beneficial. In any case, it should be clear that the State does not choose an economic model; if it happens that there only exists one economic model capable of providing software which provides the basic guarantee of these principles, this is because of historical circumstances, not because of an arbitrary choice of a given model.

Your letter continues: "4. "Your proposal imposes the use of open source software without considering the risks this carries to security, warranty, and possible violation of the intellectual property rights of third parties."

Alluding in an abstract way to "the risks this carries ", without specifically mentioning a single one of these supposed dangers, shows at the least some lack of knowledge of the topic. So, allow me to enlighten you on these points.

On security:

National security has already been mentioned in general terms in the initial discussion of the basic principles of the bill. In more specific terms, relative to the security of the software itself, it is well known that all software (whether proprietary or free) contains errors or *bugs* (in programmers' slang). But it is also well-known that the *bugs* in free software are fewer, and are fixed much more quickly, than in proprietary software. It is not in vain that numerous public bodies responsible for the IT security of State systems in developed countries require the use of free software for the same conditions of security and efficiency.

What is impossible to prove is that proprietary software is more secure than free, without the public and open inspection of the scientific community and users in general. This demonstration is impossible because the model of proprietary software itself prevents this analysis, so that any guarantee of security is based only on promises of good intentions (biased, by any reckoning) made by the producer itself, or its contractors.

It should be remembered that in many cases, the licensing conditions include de *non-disclosure* clauses which prevent the user from publicly revealing security flaws found in the licensed proprietary product.

In respect of the guarantee:

As you know perfectly well, or could find out by reading the *End User License Agreement* of the products you license, in the great majority of cases the guarantees are limited to replacement of the storage medium in case of defects, but in no case is compensation given for direct or indirect damages, loss of profits, etc... If as a result of a security *bug* in one of your products, not fixed in time by yourselves, an attacker managed to compromise crucial State systems, what guarantees, reparations and compensation would your company make in accordance with your licensing conditions? The guarantees of proprietary software, inasmuch as programs are delivered *as is*, that is, in the state in which they are, with no additional responsibility of the provider in respect of function, in no way differ from those normal with free software.

On Intellectual Property:

Questions of intellectual property fall outside the scope of this bill, since they are covered by other specific laws. The free software model in no way implies ignorance of these laws, and in fact the great majority of free software is covered by *copyright*. In reality, the inclusion of this question in your observations shows your confusion in respect of the legal framework in which free software is developed. The inclusion of the intellectual property of others in works claimed as one's own is not a practice that has been noted in the free software community; whereas, unfortunately, it has been in the area of proprietary software. As an example, the condemnation by the Commercial Court of Nanterre, France, on 27th September 2001 of Microsoft Corp. to a penalty of 3 million francs in damages and interest, for violation of intellectual property (piracy, to use the unfortunate term that your firm commonly uses in its publicity).

You go on to say that: 5. "It erroneously assumes that *open source software* is free software, that is, without cost, and therefore arrives at incorrect conclusions about money saved by the State. It has no cost-benefit analysis to back up this assumption."

This observation is wrong; in principle, freedom and lack of cost are orthogonal concepts: there is software which is proprietary and charged for (for example, MS Office), software which is proprietary and free of charge (MS Internet Explorer), software which is free and charged for (Red Hat, SuSE etc., Gnu/Linux distributions), software which is free and not charged for (Apache, OpenOffice, Mozilla), and even software which can be licensed in a range of combinations (MySQL).

Certainly free software is not necessarily free of charge. And the text of the bill does not state that it has to be so, as you will have noted after reading it. The definitions included in the bill state clearly *what* should be considered free software, at no point referring to freedom from charges. Although the possibility of savings in payments for proprietary software licenses are mentioned, the foundations of the bill clearly refer to the fundamental guarantees to be preserved and to the stimulus to local technological development. Given that a democratic state must support these principles, it has no other choice than to use software with publicly available source code, and to exchange information only in standard formats.

If the State does not use software with these characteristics, it will be weakening basic republican principles. Luckily, free software also implies lower total costs; however, even given the hypothesis (easily disproved) that it was more expensive than proprietary software, the simple existence of an effective free software tool for a particular IT function would oblige the State to use it; not by command of this bill, but because of the basic principles we enumerated at the start, and which arise from the very essence of the lawful democratic State.

You continue: 6. "It is wrong to think that open source software is free. Research by the Gartner Group (an important market researcher in the technology world, well-known worldwide) has shown that the cost of software acquisition (operating system and applications) is only 8% of the total cost of ownership that enterprises and organisations must face as a consequence of the rational and productive use of technology. "The other 92% consists of: installation, training, support, maintenance, management and administration, and downtime."

This argument repeats that already given in paragraph 5 and partly contradicts paragraph 3. For the sake of brevity we refer to the comments on those paragraphs. However, allow me to point out that your conclusion is logically false: even if according to the Gartner Group the cost of software is on average only 8% of the total cost of use, this does not in any way deny the existence of software which is free of charge, that is, with a licensing cost of zero.

In addition, in this paragraph you correctly point out that the service components and losses due to down time make up the largest part of the total cost of software use, which, as you will note, contradicts your statement regarding the small value of services suggested in paragraph 3. Now the use of free software contributes significantly to reduce the remaining life-cycle costs. This reduction in the costs of installation, support etc. can be noted in several areas: in the first place, the competitive service model of free software, support and maintenance for which can be freely contracted out to a range of suppliers competing on the grounds of quality and low cost (this is true for installation, enabling, and support, and in large part for maintenance). In the second place, due to the reproductive characteristics of the model, maintenance carried out for an application is easily replicable, without incurring large costs (that is,

without paying more than once for the same thing) since modifications, if one wishes, can be incorporated in the common fund of knowledge. Thirdly, the huge costs caused by non-functioning software (*blue screens of death*, malicious code such as virus, *worms*, and trojans, exceptions, general protection faults and other well-known problems) are reduced considerably by using more stable software. And it is well-known that one of the most remarkable virtues of free software is its stability.

You further state that: 7. "One of the arguments supporting your proposal is the supposed cheapness of open source software when compared to commercial software, without considering the possibility of volume licensing models, which can be highly advantageous for the State, as has happened in other countries."

I have already pointed out that what is in question is not the cost of the software but the principles of freedom of information, accessibility, and security. These arguments have been covered extensively in the preceding paragraphs to which I would refer you.

On the other hand, there certainly exist types of volume licensing (although unfortunately proprietary software does not satisfy the basic principles). But as you correctly pointed out in the immediately preceding paragraph of your letter, they only manage to reduce the impact of a component which makes up no more than 8% of the total.

You continue: 8. "Additionally, the approach chosen by your project (i) is clearly more expensive because of the high costs of migration; (ii) risks loss of interoperability among information systems, both inside the State and between the State and the public sector, due to the many different distributions of open source software on the market."

Let us analyse your statement in two parts. Your first argument, that migration implies high costs, is in reality an argument in favour of the bill. Because the more time goes by, the more difficult migration to another technology will become; and at the same time, the security risks associated with proprietary software will continue to increase. In this way, the use of proprietary systems and formats will make the State ever more dependent on specific suppliers. On the contrary, once a policy of using free software has been established (which certainly, does imply some cost) then on the contrary migration from one system to another becomes very simple, since all data is stored in open formats. On the other hand, migration to an open software context implies no more costs than migration between two different proprietary software contexts, which invalidates your argument completely.

The second argument refers to "loss of interoperability among information systems, both inside the State and between the State and the private sector". This statement implies a certain lack of knowledge of the way in which free

software is built, which does not maximise the dependence of the user on a particular platform, as normally happens in the realm of proprietary software. Even when there are multiple free software distributions, and numerous programs which can be used for the same function, interoperability is guaranteed as much by the use of standard formats, as required by the bill, as by the possibility of creating interoperable software given the availability of the source code.

You then say that: 9. "In most cases, open source software does not offer adequate levels of service to achieve better productivity by its users, nor does it offer warranties from well-known manufacturers. These things have caused many public entities to go back on their decisions to use open source software; they are now using commercial software in its place."

This observation is without foundation. In respect of the guarantee, your argument was rebutted in the response to paragraph 4. In respect of support services, it is possible to use free software without them (just as also happens with proprietary software), but anyone who does need them can obtain support separately, whether from local firms or from international corporations, again just as in the case of proprietary software.

On the other hand, it would contribute greatly to our analysis if you could inform us about free software projects *established* in public bodies which have already been abandoned in favour of proprietary software. We know of a good number of cases where the opposite has taken place, but do not know of any where what you describe has taken place.

You continue by observing that: 10. "This project discourages creativity in the Peruvian software industry, which sells USD 40 million worth of goods every year, USD 4 million of that exported (10th place in the ranking of Peruvian non traditional exports, more than handcrafted goods) and is a source of highly skilled jobs. With a law encouraging the use of open source software, programmers lose their intellectual property rights and their most important source of remuneration."

It is clear enough that nobody is forced to commercialise their code as free software. The only thing to take into account is that if it is not free software, it cannot be sold to the public sector. This is not in any case the main market for the national software industry. We covered some questions referring to the influence of the bill on the generation of employment which would be both highly technically qualified and in better conditions for competition above, so it seems unnecessary to insist on this point.

What follows in your statement is incorrect. On the one hand, no author of free software loses his intellectual property rights, unless he expressly wishes to place his work in the public domain. The free software movement has always been very respectful of intellectual property, and has generated widespread

public recognition of authors. Names like those of Richard Stallman, Linus Torvalds, Guido van Rossum, Larry Wall, Miguel de Icaza, Andrew Tridgell, Theo de Raadt, Andrea Arcangeli, Bruce Perens, Darren Reed, Alan Cox, Eric Raymond, and many others, are recognised world-wide for their contributions to the development of software that is used today by millions of people throughout the world, whilst there are many material authors of excellent pieces of proprietary software who remain anonymous. On the other hand, to say that the rewards for authors rights make up the main source of payment of Peruvian programmers is in any case a guess, in particular since there is no proof to this effect, nor a demonstration of how the use of free software by the State would influence these payments.

You go on to say that: 11. "Since open source software can be freely distributed, it cannot make any money for its developers by exportation. In this way, it weakens the multiplier effect of software sales to other countries and stunts the growth of this local industry, which the State should be stimulating."

This statement shows once again complete ignorance of the mechanisms of and market for free software. It tries to claim that the market of sale of non-exclusive rights for use (sale of licences) is the only possible one for the software industry, when you yourself pointed out several paragraphs above that it is not even the most important one. The incentives that the bill offers for the growth of a supply of better qualified professionals, together with the increase in experience that working on a large scale with free software within the State will bring for Peruvian technicians, will place them in a highly competitive position to offer their services abroad.

You then state that: "12. In the forum, the importance of the use of open source software in education was discussed, without commenting on the complete failure of this initiative in countries like Mexico. There, the same State officials who supported the project now say that open source software did not provide a learning experience to children in schools, adequate levels of training were not available nationwide, inadequate support for the platform was provided, and the software was not integrated well enough with existing school computer systems."

In fact Mexico has gone into reverse with the Red Escolar (Schools Network) project. This is due precisely to the fact that the driving forces behind the Mexican project used license costs as their main argument, instead of the other reasons specified in our project, which are far more essential. Because of this conceptual mistake, and as a result of the lack of effective support from the SEP (Secretary of State for Public Education), the assumption was made that to implant free software in schools it would be enough to drop their software budget and send them a CD ROM with GNU/Linux instead. Of course this failed, and it could not have been otherwise, just as school laboratories fail when they use proprietary software and have no budget for implementation and maintenance. That is exactly why our bill is not limited to making the use

of free software mandatory, but recognises the need to create a viable migra-
tion plan, in which the State undertakes the technical transition in an orderly
way in order to then enjoy the advantages of free software.

You end with a rhetorical question: 13. "If open source software fulfils all the
requirements of State entities, why should a law be needed to adopt its use?
Should not the market freely choose which products provide more benefits
and value?"

We agree that in the private sector of the economy, it must be the market that
decides which products to use, and no State interference is permissible there.
However, in the case of the public sector, the reasoning is not the same: as
we have already established, the State archives, handles, and transmits infor-
mation which does not belong to it, but which is entrusted to it by citizens,
who have no alternative under the rule of law. As a counterpart to this legal
requirement, the State must take extreme measures to safeguard the integrity,
confidentiality, and accessibility of this information. The use of proprietary
software raises serious doubts as to whether these requirements can be ful-
filled, lacks conclusive evidence in this respect, and so is not suitable for use
in the public sector.

The need for a law is based, firstly, on the realisation of the fundamental prin-
ciples listed above in the specific area of software; secondly, on the fact that
the State is not an ideal homogeneous entity, but made up of multiple bodies
with varying degrees of autonomy in decision making. Given that it is inap-
propriate to use proprietary software, the fact of establishing these rules in
law will prevent the personal discretion of any State employee from putting at
risk the information which belongs to citizens. And above all, because it con-
stitutes an up-to-date reaffirmation in relation to the means of management
and communication of information used today, it is based on the republican
principle of openness to the public.

In conformance with this universally accepted principle, the citizen has the
right to know all information held by the State and not covered by well-found-
ed declarations of secrecy based on law. Now, software deals with information
and is itself information. Information in a special form, capable of being in-
terpreted by a machine in order to execute actions, but crucial information all
the same because the citizen has a legitimate right to know, for example, how
his vote is computed or his taxes calculated. And for that he must have free
access to the source code and be able to prove to his satisfaction the programs
used for electoral computations or calculation of his taxes.

D.5. Decree of Measures to Promote the Knowledge Society in Andalucía

Below are some of the articles, related to free software, of the abovementioned
Decree on Measures to Encourage the Knowledge Society in Andalucía [99].

- Article 11. Educational materials in computer format.

  1. All public teaching centres will have educational materials and programs in computerised format, preferably based on free software. In any case, the centres will receive these formats from the Regional Government of Andalucía.

  2. Likewise, the teachers will receive incentives for using computerised curricular materials and programs or using Internet, especially with regard to developments made using free software.

- Article 31. Free software.

  1. When purchasing computer equipment that will be used in public teaching centres for educational activities, it should be ensured that all the hardware is compatible with operating systems based on free software. Computers will come preinstalled with all the free software that is necessary for the specific purposes for which they are intended.

  2. The computer equipment that the Regional Government of Andalucía provides for public access to Internet will be based on free software products.

  3. The Regional Government of Andalucía will foster the dissemination and the personal, domestic and educational use of free software. For these purposes, an online advice service will be established for the installation and use of these types of products.

- Article 49. Objective.

  1. There will be subsidies for the development of innovative projects that facilitate the integration of IT and communications in professional and occupational training.

  2. These projects will follow one of the following models:

  a) Preparation of materials and contents of professional and occupational training for their use and dissemination by Internet, especially with regard to the developments made using free software.

  b) Training initiatives using innovative methods, such as long-distance learning and methods whereby the students only need to attend the courses personally on certain occasions.

5. Appendix E. Creative Commons' Attribution-ShareAlike

Version 3.0 Unported

1. License

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORISED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. TO THE EXTENT THIS LICENSE MAY BE CONSIDERED TO BE A CONTRACT, THE LICENSOR GRANTS YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS.

1. Definitions

a) "**Adaptation**" means a work based upon the Work, or upon the Work and other pre-existing works, such as a translation, adaptation, derivative work, arrangement of music or other alterations of a literary or artistic work, or phonogram or performance and includes cinematographic adaptations or any other form in which the Work may be recast, transformed, or adapted including in any form recognisably derived from the original, except that a work that constitutes a Collection will not be considered an Adaptation for the purpose of this License. For the avoidance of doubt, where the Work is a musical work, performance or phonogram, the synchronisation of the Work in timed-relation with a moving image ("synching") will be considered an Adaptation for the purpose of this License.

b) "**Collection**" means a collection of literary or artistic works, such as encyclopedias and anthologies, or performances, phonograms or broadcasts, or other works or subject matter other than works listed in Section 1(f) be-

low, which, by reason of the selection and arrangement of their contents, constitute intellectual creations, in which the Work is included in its entirety in unmodified form along with one or more other contributions, each constituting separate and independent works in themselves, which together are assembled into a collective whole. A work that constitutes a Collection will not be considered an Adaptation (as defined below) for the purposes of this License.

c) **"Creative Commons Compatible License"** means a license that is listed at http://creativecommons.org/compatiblelicenses that has been approved by Creative Commons as being essentially equivalent to this License, including, at a minimum, because that license: (i) contains terms that have the same purpose, meaning and effect as the License Elements of this License; and, (ii) explicitly permits the relicensing of adaptations of works made available under that license under this License or a Creative Commons jurisdiction license with the same License Elements as this License.

d) **"Distribute"** means to make available to the public the original and copies of the Work or Adaptation, as appropriate, through sale or other transfer of ownership.

e) **"License Elements"** means the following high-level license attributes as selected by Licensor and indicated in the title of this License: Attribution, ShareAlike.

f) **"Licensor"** means the individual, individuals, entity or entities that offer(s) the Work under the terms of this License.

g) **"Original Author"** means, in the case of a literary or artistic work, the individual, individuals, entity or entities who created the Work or if no individual or entity can be identified, the publisher; and in addition (i) in the case of a performance the actors, singers, musicians, dancers, and other persons who act, sing, deliver, declaim, play in, interpret or otherwise perform literary or artistic works or expressions of folklore; (ii) in the case of a phonogram the producer being the person or legal entity who first fixes the sounds of a performance or other sounds; and, (iii) in the case of broadcasts, the organisation that transmits the broadcast.

h) **"Work"** means the literary and/or artistic work offered under the terms of this License including without limitation any production in the literary, scientific and artistic domain, whatever may be the mode or form of its expression including digital form, such as a book, pamphlet and other writing; a lecture, address, sermon or other work of the same nature; a dramatic or dramatico-musical work; a choreographic work or entertainment in dumb show; a musical composition with or without words; a cinematographic work to which are assimilated works expressed by a process analogous to cinematography; a work of drawing, painting, architecture,

sculpture, engraving or lithography; a photographic work to which are as-similated works expressed by a process analogous to photography; a work of applied art; an illustration, map, plan, sketch or three-dimensional work relative to geography, topography, architecture or science; a performance; a broadcast; a phonogram; a compilation of data to the extent it is pro-tected as a copyrightable work; or a work performed by a variety or circus performer to the extent it is not otherwise considered a literary or artistic work.

i) "**You**" means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.

j) "**Publicly Perform**" means to perform public recitations of the Work and to communicate to the public those public recitations, by any means or process, including by wire or wireless means or public digital perfor-mances; to make available to the public Works in such a way that members of the public may access these Works from a place and at a place individ-ually chosen by them; to perform the Work to the public by any means or process and the communication to the public of the performances of the Work, including by public digital performance; to broadcast and rebroad-cast the Work by any means including signs, sounds or images.

k) "**Reproduce**" means to make copies of the Work by any means including without limitation by sound or visual recordings and the right of fixation and reproducing fixations of the Work, including storage of a protected performance or phonogram in digital form or other electronic medium.

2. Fair Dealing Rights. Nothing in this License is intended to reduce, limit, or restrict any uses free from copyright or rights arising from limitations or exceptions that are provided for in connection with the copyright protection under copyright law or other applicable laws.

3. License Grant. Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

a) to Reproduce the Work, to incorporate the Work into one or more Collec-tions, and to Reproduce the Work as incorporated in the Collections;

b) to create and Reproduce Adaptations provided that any such Adaptation, including any translation in any medium, takes reasonable steps to clearly label, demarcate or otherwise identify that changes were made to the orig-inal Work. For example, a translation could be marked "The original work

was translated from English to Spanish," or a modification could indicate "The original work has been modified.";

c) to Distribute and Publicly Perform the Work including as incorporated in Collections; and,

d) to Distribute and Publicly Perform Adaptations.

e) For the avoidance of doubt:

    a) **Non-waivable Compulsory License Schemes**. In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme cannot be waived, the Licensor reserves the exclusive right to collect such royalties for any exercise by You of the rights granted under this License;

    b) **Waivable Compulsory License Schemes**. In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme can be waived, the Licensor waives the exclusive right to collect such royalties for any exercise by You of the rights granted under this License; and,

    c) **Voluntary License Schemes**. The Licensor waives the right to collect royalties, whether individually or, in the event that the Licensor is a member of a collecting society that administers voluntary licensing schemes, via that society, from any exercise by You of the rights granted under this License.

The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats. Subject to Section 8(f), all rights not expressly granted by Licensor are hereby reserved.

4. Restrictions. The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

a) You may Distribute or Publicly Perform the Work only under the terms of this License. You must include a copy of, or the Uniform Resource Identifier (URI) for, this License with every copy of the Work You Distribute or Publicly Perform. You may not offer or impose any terms on the Work that restrict the terms of this License or the ability of the recipient of the Work to exercise the rights granted to that recipient under the terms of the License. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties with every copy of the Work You Distribute or Publicly Perform. When You Distribute or Publicly Perform the Work, You may not impose any effective technological measures on the Work that restrict the ability of a recipient

of the Work from You to exercise the rights granted to that recipient under the terms of the License. This Section 4(a) applies to the Work as incorporated in a Collection, but this does not require the Collection apart from the Work itself to be made subject to the terms of this License. If You create a Collection, upon notice from any Licensor You must, to the extent practicable, remove from the Collection any credit as required by Section 4(c), as requested. If You create an Adaptation, upon notice from any Licensor You must, to the extent practicable, remove from the Adaptation any credit as required by Section 4(c), as requested.

b) You may Distribute or Publicly Perform an Adaptation only under the terms of: (i) this License; (ii) a later version of this License with the same License Elements as this License; (iii) a Creative Commons jurisdiction license (either this or a later license version) that contains the same License Elements as this License (e.g., Attribution-ShareAlike 3.0 US)); (iv) a Creative Commons Compatible License. If you license the Adaptation under one of the licenses mentioned in (iv), you must comply with the terms of that license. If you license the Adaptation under the terms of any of the licenses mentioned in (i), (ii) or (iii) (the "Applicable License"), you must comply with the terms of the Applicable License generally and the following provisions: (I) You must include a copy of, or the URI for, the Applicable License with every copy of each Adaptation You Distribute or Publicly Perform; (II) You may not offer or impose any terms on the Adaptation that restrict the terms of the Applicable License or the ability of the recipient of the Adaptation to exercise the rights granted to that recipient under the terms of the Applicable License; (III) You must keep intact all notices that refer to the Applicable License and to the disclaimer of warranties with every copy of the Work as included in the Adaptation You Distribute or Publicly Perform; (IV) when You Distribute or Publicly Perform the Adaptation, You may not impose any effective technological measures on the Adaptation that restrict the ability of a recipient of the Adaptation from You to exercise the rights granted to that recipient under the terms of the Applicable License. This Section 4(b) applies to the Adaptation as incorporated in a Collection, but this does not require the Collection apart from the Adaptation itself to be made subject to the terms of the Applicable License.

c) If You Distribute, or Publicly Perform the Work or any Adaptations or Collections, You must, unless a request has been made pursuant to Section 4(a), keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilising: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or if the Original Author and/or Licensor designate another party or parties (e.g., a sponsor institute, publishing entity, journal) for attribution ("Attribution Parties") in Licensor's copyright notice, terms of service or by other reasonable means, the name of such party or parties; (ii) the title of the Work if supplied; (iii) to the extent reasonably practicable, the URI, if any, that Licensor specifies

to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work; and (iv), consistent with Section 3(b), in the case of an Adaptation, a credit identifying the use of the Work in the Adaptation (e.g., "French translation of the Work by Original Author," or "Screenplay based on original Work by Original Author"). The credit required by this Section 4(c) may be implemented in any reasonable manner; provided, however, that in the case of a Adaptation or Collection, at a minimum such credit will appear, if a credit for all contributing authors of the Adaptation or Collection appears, then as part of these credits and in a manner at least as prominent as the credits for the other contributing authors. For the avoidance of doubt, You may only use the credit required by this Section for the purpose of attribution in the manner set out above and, by exercising Your rights under this License, You may not implicitly or explicitly assert or imply any connection with, sponsorship or endorsement by the Original Author, Licensor and/or Attribution Parties, as appropriate, of You or Your use of the Work, without the separate, express prior written permission of the Original Author, Licensor and/or Attribution Parties.

d) Except as otherwise agreed in writing by the Licensor or as may be otherwise permitted by applicable law, if You Reproduce, Distribute or Publicly Perform the Work either by itself or as part of any Adaptations or Collections, You must not distort, mutilate, modify or take other derogatory action in relation to the Work which would be prejudicial to the Original Author's honor or reputation. Licensor agrees that in those jurisdictions (e.g. Japan), in which any exercise of the right granted in Section 3(b) of this License (the right to make Adaptations) would be deemed to be a distortion, mutilation, modification or other derogatory action prejudicial to the Original Author's honor and reputation, the Licensor will waive or not assert, as appropriate, this Section, to the fullest extent permitted by the applicable national law, to enable You to reasonably exercise Your right under Section 3(b) of this License (right to make Adaptations) but not otherwise.

5. Representations, Warranties and Disclaimer

UNLESS OTHERWISE MUTUALLY AGREED TO BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTIBILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

6. Limitation on Liability. EXCEPT TO THE EXTENT REQUIRED BY APPLI-CABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. Termination

a) This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Adaptations or Collections from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.

b) Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

a) Each time You Distribute or Publicly Perform the Work or a Collection, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.

b) Each time You Distribute or Publicly Perform an Adaptation, Licensor offers to the recipient a license to the original Work on the same terms and conditions as the license granted to You under this License.

c) If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

d) No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.

e) This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements

or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.

f) The rights granted under, and the subject matter referenced, in this License were drafted utilising the terminology of the Berne Convention for the Protection of Literary and Artistic Works (as amended on September 28, 1979), the Rome Convention of 1961, the WIPO Copyright Treaty of 1996, the WIPO Performances and Phonograms Treaty of 1996 and the Universal Copyright Convention (as revised on July 24, 1971). These rights and subject matter take effect in the relevant jurisdiction in which the License terms are sought to be enforced according to the corresponding provisions of the implementation of those treaty provisions in the applicable national law. If the standard suite of rights granted under applicable copyright law includes additional rights not granted under this License, such additional rights are deemed to be included in the License; this License is not intended to restrict the license of any rights under applicable law.

6. Appendix F. GNU Free Documentation License

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- **A.** Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

- **B.** List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

- **C.** State on the Title page the name of the publisher of the Modified Version, as the publisher.

- **D.** Preserve all the copyright notices of the Document.

- **E.** Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

- **F.** Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

- **G.** Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

- **H.** Include an unaltered copy of this License.

- **I.** Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

- **J.** Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

- **K.** For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

- **L.** Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

- **M.** Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

- **N.** Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

- **O.** Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as in-

variant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties - for example, statements of peer review or that the text has been approved by an organisation as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document 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.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation 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. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

WHAT IS FREE SOFTWARE? WHAT IS IT AND WHAT ARE THE IMPLICATIONS OF A FREE PROGRAM LICENCE? HOW IS FREE SOFTWARE DEVELOPED? HOW ARE FREE SOFTWARE PROJECTS FINANCED AND WHAT ARE THE BUSINESS MODELS ASSOCIATED TO THEM THAT WE ARE EXPERIENCING? WHAT MOTIVATES DEVELOPERS, ESPECIALLY VOLUNTEERS, TO BECOME INVOLVED IN FREE SOFTWARE PROJECTS? WHAT ARE THESE DEVELOPERS LIKE? HOW ARE THEIR PROJECTS COORDINATED, AND WHAT IS THE SOFTWARE THAT THEY PRODUCE LIKE? IN SHORT, WHAT IS THE OVERALL PANORAMA OF FREE SOFTWARE?

THESE ARE THE SORT OF QUESTIONS THAT WE WILL TRY TO ANSWER IN THIS DOCUMENT. BECAUSE ALTHOUGH FREE SOFTWARE IS INCREASING ITS PRESENCE IN THE MEDIA AND IN DEBATES AMONG IT PROFESSIONALS, AND ALTHOUGH EVEN CITIZENS IN GENERAL ARE STARTING TO TALK ABOUT IT, IT IS STILL UNKNOWN FOR MANY PEOPLE. AND EVEN THOSE WHO ARE FAMILIAR WITH IT ARE OFTEN AWARE OF JUST SOME OF ITS FEATURES, AND MOSTLY IGNORANT ABOUT OTHERS.