Open Source in Higher Education: Case Study Computer Science at the University of Fribourg

Amos Brocco and Fulvio Frapolli

February 14, 2011

Contents

Chapter 1

Introduction

Firefox, Openoffice, GIMP, etc. these are the names of just few of the open source software tools that are currently used by millions of people around the world. While most of the users neglect or ignore the fact that these tools are open, these applications surely owe part of their success to the fact that people have been able to freely work, adapt, and improve them thanks to the open source distribution philosophy. In this paper we want to highlight the benefits of open solutions in education and promote the active use of open source software in computer science classes.

1.1 About this document

This work is the product of a team work between Amos Brocco and Fulvio Frapolli as the result of the tight collaboration during these last years at both the scientific research and teaching levels. Being consumers of both open source software and content in our private life we tried to promote open source solutions also during the computer science classes that we were responsible for. We exhibit here thus a single document which has been organized so that an equal amount of work has been provided by each author.

1.2 Open source

Computers are very complex machines that can understand only a very simple code, namely the binary code (zeroes and ones). Developers don't write applications using binary code, but instead rely on higher level programming languages which are, compared to machine code, human readable. The instructions given to the computer in a program determine what the processor and other devices will do. A special tool, called *compiler*, translates these instructions into binary machine code, so that the application can be understood and executed by the computer. The ensemble of all the files that contain those instruction is called the *source code* of an application. When an application is released according to the proprietary *closed source* model, only the binary representation is distributed, not the source code. This means that for a human it is almost impossible to understand what the application really does, or to modify the application to better fit his or her needs. Moreover, the copyright terms forbid to redistribute the application to other people (either as-is or modified).

The term *open source* is referred to the source code of a computer program. In open source applications the source code is distributed along with the application (or is available to the end-user), meaning that anyone can study its contents and modify it. Moreover, the programmer gives explicit permission to redistribute copies of the application, provided that some rules are obeyed. Open source software is thus very different compared to proprietary solutions: whereas the former promote transparency and sharing, the latter are typically obscure and protected.

1.3 Open source and education

The goal of open source solutions in the context of education, and in particular in computer science, is twofold: on one hand to provide the student with, generally free, tools for crating, modifying and exchanging information; on the other hand, and for what concerns computer science education, to allow the teacher to explain the fundamentals of computer science in a real world scenario. From a more general point of view, opening up education increases the availability of learning material, so that both teachers and students can (quoting Newton)

> see further (. . .) by standing on the shoulders of giants

.

In this paper we give an insight on the use of open source solutions and of open knowledge in education. In order to achieve our goal, several concerns are to be treated. Accordingly this document is divided in three parts. In the first part, a definition of the main open source concepts is provided. Furthermore, a survey of existing open source solutions in the context of higher education is presented. In the second part the benefits of openness are discussed, and the different beneficial factors are presented. Finally, open tools and solutions, deployment opportunities for educational purposes, and a case study concerning the use of open source in a computer science class at the University of Fribourg is the topic of the third part. More specifically, we describe how open source has been employed in our department and draw some conclusions on its benefits and drawbacks from our personal experience.

Chapter 2

An open information society

In recent years, we witnessed an explosion in the amount of content produced, distributed and shared by individuals, in different formats, such as texts, music or videos. Spurred by the birth of content-sharing and social-networking websites, previous technical limitations that could have hindered casual users from distributing their productions have ceased to exist. In this context, open access policies aim at easing the legal provisions that may prevent free distribution and fruition of information. Because scientific research and the evolution of common knowledge depend on sharing results and findings amongst people, leveraging *openness* at all levels of content production and distribution brings enormous benefits to the community.

2.1 What is open?

Before digging into the different categories of open content, a precise definition of what should be considered as *open* is necessary. As pointed out in [22], the success of open solutions depends on several factors, one of them being a common agreement on the definition of openness. It is important to differentiate between various terms that are often misunderstood or abused when referring to content that can be easily found on the web. Because content can be understood as *open* at different degrees, to fully consider the implications of *openness* we first report the definition given in [20], namely

> A piece of knowledge is open if you are free to use, reuse, and redistribute it subject only, at most, to the requirement to attribute and share-alike.

A common fallacy in understanding this definition concerns the meaning of *free*, which does not refer to the price or cost but rather, as pointed out in [30], to *freedom*; to quote Richard Stallman, *Think free as in free speech, not free beer*. In this regard, open content could be made available for a fee, although this is rarely the case. On the contrary, freedom of reuse and redistribute cannot involve any fee or royalty: put it simply, if Alice obtains some open content from Bob, and redistributes it to Tom, Bob cannot exercise any royalty right.

Another take on the definition of *openness* is offered by the *Open Source Definition* [21], which refers to computer software and cites different criteria such as: free redistribution, source code availability, right to modify and produce derived works, etc.

Although the freedom given to open content offers a wider range of opportunities, licenses have been created to precisely define the rights given to the end-users. Creators of open content retain full rights over their productions, and copyright is preserved. Hence, redistribution licences are not meant as an alternative to copyright, but aim at giving additional permissions and rights to everyone within the distribution chain, up to end-users.

In respect to the previous definitions, a large number of works cannot be clearly classified as open. In particular, some of the information found on the Internet or commercial software, even though it can be freely accessed or used is not open following the aforementioned definitions. Notable examples include popular newspaper websites, that provide freely accessible content but do not grant any right to reproduce or reuse it, and *freeware* software such as Skype or the Opera web browser. It is noteworthy to mention that specific redistribution schemes might be still agreed upon to obtain the right to reproduce content (typically by paying a redistribution license), but opposed to open content, such grant is normally not transferable and does not extend to end-users.

2.2 Copyright versus Ownership

An important clarification should be made concerning the purpose of copyright and the concept of ownership. Ownership typically pertains to physical goods (for example, books or audio tapes), but also to abstract ones such as royalty rights to novels and songs. Because abstract goods usually require a physical distribution medium, copyright laws have been created to draw a line between physical ownership of this medium and that of the content. More specifically, while a person can acquire the ownership of a music compact disc, the rights and ownership of its contents (i.e. the songs) remain to the composer. As there is practically no law limiting physical ownership, the focus of openness must be put on content distribution and copyright laws. While current laws do a very good job at protecting authors' rights, they impose serious limitations to the end-user. In particular they prevent any redistribution of the content.

Ownership of intangible content is traditionally referred to as *intellectual property* (IP), and groups several concerns such as copyright, trademarking, patents, etc. There exists an organization that is devoted to the protection of such *property* (the World Intellectual Property Organization (WIPO) established in 1967), that advertises IP as a financial incentive fostering research and development. While protection of ideas could really push economic growth in certain areas (such as industrial processes), there is a common criticism ([7, 27]) arguing that in recent times and in several areas (such as computer software) the opposite happens, with excessive protection limiting innovation. Accordingly, a review of what is really to be protected is necessary, but pressure from large companies hinders this process.

2.3 Copyright and Licenses

Openness aims at increasing the distribution and redistribution rights given by the authors, not by replacing copyright, but by leveraging existing copyright laws. As content authors retain full ownership of their product, distribution licenses can be legally enforced to either broaden the end-user distribution rights, or even restrict them.

Several licenses exists to govern the distribution of content ranging from computer software to artistic productions. One of the first, and most known open licenses is the **General Public License** (GPL) [11], proposed by Richard Stallman in 1984 to regulate the distribution of a free operating system called GNU [9]. The GPL is also an example of *copyleft* license, which groups all distribution licenses that give the right to distribute modified or unmodified versions of a work provided that the same rights are preserved. Accordingly, the GPL is referred to as a *non-permissive* license. On the contrary, *permissive* licenses allow for narrower redistribution terms on derivative work, and examples include the **BSD** [24] or the **MIT** [19] licenses: the recipient of software distributed under such licenses can choose to incorporate it within non-free or proprietary software. Figure 2.1 illustrates the differences between several licenses used for the distribution of open source software.

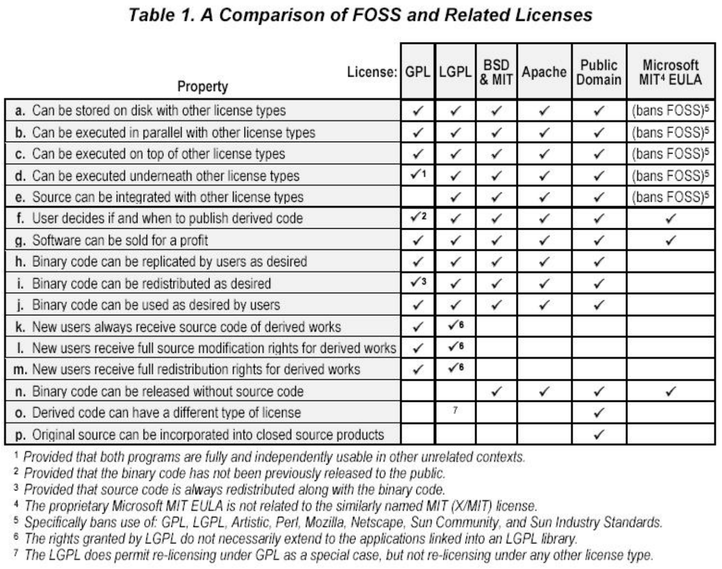


Table 1. A Comparison of FOSS and Related Licenses

Property / License:	GPL	LGPL	BSD & MIT	Apache	Public Domain	Microsoft MIT[4] EULA
a. Can be stored on disk with other license types	✓	✓	✓	✓	✓	(bans FOSS)[5]
b. Can be executed in parallel with other license types	✓	✓	✓	✓	✓	(bans FOSS)[5]
c. Can be executed on top of other license types	✓	✓	✓	✓	✓	(bans FOSS)[5]
d. Can be executed underneath other license types	✓[1]	✓	✓	✓	✓	(bans FOSS)[5]
e. Source can be integrated with other license types		✓	✓	✓	✓	(bans FOSS)[5]
f. User decides if and when to publish derived code	✓[2]	✓	✓	✓	✓	✓
g. Software can be sold for a profit	✓	✓	✓	✓	✓	✓
h. Binary code can be replicated by users as desired	✓	✓	✓	✓	✓	
i. Binary code can be redistributed as desired	✓[3]	✓	✓	✓	✓	
j. Binary code can be used as desired by users	✓	✓	✓	✓	✓	
k. New users always receive source code of derived works	✓	✓[6]				
l. New users receive full source modification rights for derived works	✓	✓[6]				
m. New users receive full redistribution rights for derived works	✓	✓[6]				
n. Binary code can be released without source code			✓	✓	✓	✓
o. Derived code can have a different type of license		[7]			✓	
p. Original source can be incorporated into closed source products					✓	

[1] *Provided that both programs are fully and independently usable in other unrelated contexts.*
[2] *Provided that the binary code has not been previously released to the public.*
[3] *Provided that source code is always redistributed along with the binary code.*
[4] *The proprietary Microsoft MIT EULA is not related to the similarly named MIT (X/MIT) license.*
[5] *Specifically bans use of: GPL, LGPL, Artistic, Perl, Mozilla, Netscape, Sun Community, and Sun Industry Standards.*
[6] *The rights granted by LGPL do not necessarily extend to the applications linked into an LGPL library.*
[7] *The LGPL does permit re-licensing under GPL as a special case, but not re-licensing under any other license type.*

Figure 2.1: Comparison of some free and open source software (FOSS) licenses (original from [5])

Whereas most open licenses have been created for use with computer software, a number of *generic* licenses have recently started to appear (GNU FDL [10], Open Audio License [8], etc.). In contrast to software licenses, free content ones concern a wider range of use cases, such as the distribution of documentation, pictures, audio, or movies. In this regard, the most famous licensing terms are the ones known as **Creative Commons** (CC) [16].

Finally, works can also get rid of copyright by being released as **Public Domain**. In this case, the author explicitly gives up its rights on it, so that the content can be freely used, modified, and distributed.

Free content and software licenses depend on the willingness of the recipients to comply with the redistribution terms, and rely on the copyright law to protect the authors rights. Accordingly, because licensing terms are often ignored once the content has been obtained or are difficult to fully understand, abuses can arise.

2.4 Abusing openness

The downside of making works available under an open license is the risk of unfair exploitation of the whole content or just part of it. Notable cases from computer software include the use of open source code within proprietary products or the failure to fully comply with the licensing terms, for example by not making the source code available as soon as the product is released [12]. In education, the risk of openness concerns plagiarism: students may be tempted to borrow free content and claim ownership over it. This issue also exists with non-free content, but the *free* nature and availability of open content is often deceiving. Because open content usually relies on the collaborative work of an engaged community, abuses can lead to negative effects on the motivation of each participant to contribute to a project. Fortunately, copyright and licenses provide an appropriate protection against such abuses. Several entities closely monitor (either *by hand* or in an automated way [25]) the produced content, and violators are typically discovered and punished.

Chapter 3

Benefits of Openness

Opening up content brings a number of advantages and freedom to end-users; unfortunately, a number of business companies are more geared towards limiting what the user can do up to the point of introducing artificial limitations such as *Digital Right Management* (DRM). In this paper we will not deal with such cases, but concentrate on a full understanding of the benefits of open solutions. Accordingly, a review of the typical communication flow of content/information between a producer and a consumer can help in understanding the differences between a proprietary and a non-proprietary situation.

Figure 3.1 depicts the typical flow between producers and consumers in the proprietary world: the information is created by some entity, represented according to some format known to both the sender and the receiver, and then transmitted to the end-user. The content is closed, and the end-user cannot transfer a derivative work to another consumer.

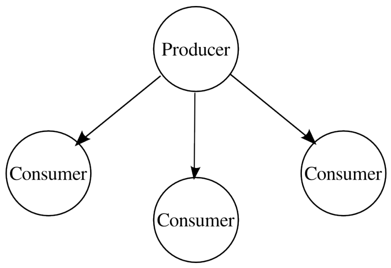


Figure 3.1: Proprietary software distribution (Example)

Figure 3.2 illustrates a typical communication in an open (non-proprietary) situation: the end-user is not only the receiver of the information, but can become an active aggregator of multiple sources and finally the producer (*prosumer*, as **pro**ducer and con**sumer**) of some derivative work that can be transferred to other users. This model results in a number of benefits [3], such as a more agile development that evolves more rapidly and quickly adapts to new environmental conditions and needs, and higher quality standards: as soon as a new feature has been added or a problem has been fixed, the whole community can take advantage.

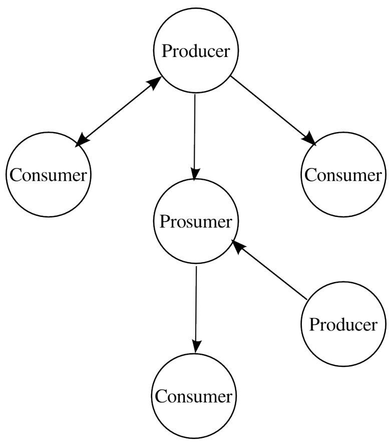


Figure 3.2: Open-source software distribution (Example)

The previous example raises a number of issues related to open content. In particular, an important component is needed to support the *open* transfer of information, namely the representation *format*. If the sender and the receiver of the information do not agree on a common communication *language*, communication itself cannot take place. In this regard, and in the light of digital information, enabling open communication relies on open formats and possibly open applications. In the next chapter examples of open formats and tools will be provided.

3.1 Success factors

There are many factors that are to be accounted for the success of the open source philosophy. In the following we identify and discuss what are, in our opinion, four main beneficial factors of openness, namely availability, cost, distributability, and community.

Availability The first benefit of opening up content is an increased fruibility of information. The right to redistribute it increases the number of providers, thus consumers have more choice and more possibilities to obtain it.

Independence from a single distributor has a number of advantages. As an example, in January 2010 the government of the United States has asked the sourceforge.net website (which is popular for hosting open source software projects) to deny access to users connecting from embargo countries such as Sudan, Syria, Iran, North Korea, and Cuba [2]. Such restriction is clearly against Section 5 of the open source definition [21], which prohibits discrimination *against any person or group of persons*. Although such restriction prevents people in some countries from downloading software from sourceforge.net, thanks to the redistribution clause of open source licenses, the very same projects could be hosted and distributed by other providers outside USA, thus bypassing the embargo restriction. When it comes to open source software, availability also concerns the source code of the application being distributed. By enabling user access to the inner working of a program, adaptation (for example to enable running the application on a different computer architecture), reuse of components, and interoperation with other programs are possible. Finally, open solutions also help survivability of digital content, as proven by the Caspar project [1] promoted by the European Union. Open formats and applications ensure that content created today would still be accessible in the future, because all the specifications on how to decode data will remain available. On the contrary, with proprietary solutions future access to information depends on the interests of the single or few entities that own the source code of the tools used to create and manage data.

Cost The choice of employing an open solution rather than a proprietary one is often related to the cost of deployment or of ownership. While it is true that most open solutions come at no-cost, we must stress that the real benefits of them should be made concerning the freedom of use rather than by looking at their price. In fact, institutions should allocate some budget even to free solutions [22]. Cost advantages also come from the fact that open solutions are typically not tied to a single vendor. As such, the end-user can choose between different offers, and select the one that best suits his needs.

Distributability Content often needs to be adapted before being used: translation, language simplification, etc. With non-free content such changes cannot be redistributed, meaning that end-users cannot benefit from adaptations made by others. Conversely, with free content the users are allowed, and encouraged to redistribute new and improved versions. The distributability factor is strongly tied with availability and the cost factor: if content is available at no cost in the first instance, re-distributability is easier as there is no cost involved.

Community In [22] the authors highlight "community building" as an essential factor for successful deployment of open source solutions. Behind every open source project stands a community of people committed to work, test, document and contribute back using their knowledge and time.

The importance of the community is very well resumed in a variation of the previously cited Newton's quote that reads:

> (...) advances are made by standing on the shoulders of giants. (...) if there are enough of you, you can advance just as far by stepping on each others toes.

The idea of sharing, improving, and contributing back is central to scientific research and progress, as all important scientific discoveries have generally been built on the results of previous works. In his "The cathedral and the bazaar" [23] essay, Eric S. Raymond notes how the development of open source software benefits from the seemingly confusing activities of developers all around the world. Moreover he mentions that "Good programmers know what to write. Great ones know what to rewrite (and reuse)", which emphasizes the freedom given by open source licenses. Several researchers have tried to map and describe the relations inside open source communities [18, 1], and such studies have shown

[1] `http://www.casparpreserves.eu/`

that people contributing to open source projects are motivated by altruism, skill improvement, and gift giving as a way of getting new ideas. As a side-effect, work done in the community is recognized and represents useful professional experience for the contributor.

Chapter 4

Open$\star$ in Education

[1]

In this section we move our focus toward open solutions in education. More specifically, we want to highlight the useful characteristics of openness in respect to the two main tasks of education: teaching and learning.

As noted in [15], open source has always been closely related to the academic world. Open source is having a great impact in driving the shift toward digital education; in particular, in this digitization we can recognize four directions, namely virtual universities, online classes, education portals, and courseware. To effectively support high-quality digital education, large investments are required. In this context, open source software offer an alternative to expensive proprietary solutions, thus providing a cost advantage.

It is nonetheless our opinion that the real benefit of open source and open content in general relies not on the cost factor but on the ideology behind it. We argue that opening up education increases the availability of high quality *ready to use* content. This has the effect of lowering the effort required to create teaching support material, leaving space to the actual *knowledge transfer*. Being able to reuse, adapt, extend and redistribute other people's work helps keeping knowledge up-to-date or suited for specific target groups. On the other hand, freely available content lowers the access barriers for students, and more appealing classes can be delivered.

Based on these considerations, we can identify four main concerns related to freedom and openness in education: making existing knowledge and content available, enabling modification of content or creation of new content, supporting knowledge and content transfer, enabling sharing and collaboration. Each of these concerns entails different requirements that need to be detailed.

The first barrier to overcome is making existing knowledge and content available. As the amount of information continues to grow, it is essential that past findings and discoveries are organized and easily accessible. Easing access to knowledge can be achieved at different levels, such as by removing the cost associated with information, by providing indexes and organized collections, by adapting the content to the consumers (according to their language, age, or education).

Whereas e-learning platforms take the necessary steps to deliver education

[1]We use the term Open$\star$ to group all the different facets of openness.

in the digital realm, open source solutions provide the necessary tools to do so, and ensure transparency and flexibility as well as increase knowledge exchange [14].

The second step concerns the modification of existing content and the creation of new works. If content cannot be modified to fulfill specific users' needs it is often useless (for example, teaching materials). Accordingly, tools for creating or modifying works need to be easily available to end-users. As the creation and modification of open content depends on such tools, open solutions to this problem are preferable.

To support the transfer of knowledge and content, open standards for representing information need to be employed. Communication can only take place if all parties share a common language, and no one has to be forbidden to learn it. In this respect, proprietary file formats may prevent a large number of users from accessing the information if the corresponding software application is not available.

Finally, to promote knowledge sharing and collaboration amongst users, specific platforms need to be deployed.

4.1 Open Knowledge and Content

The most well-known source of open knowledge on the Internet is probably Wikipedia [2]. Created in 2001 by Jimmy Wales and Larry Sanger, Wikipedia aims at enabling worldwide free access to knowledge, promoting collaborative authoring and reviewing. The project was created as a *spin-off* of a more restrictive online encyclopedia called Nupedia, which relied on expert and dependable sources for the creation of content. Wikipedia is praised for the large amount of information that can be found on it; nevertheless, critics concerning the quality of some articles have appeared. Figure 4.1 shows the evolution in the number of articles in the english Wikipedia from 2001 to 2007 [3].

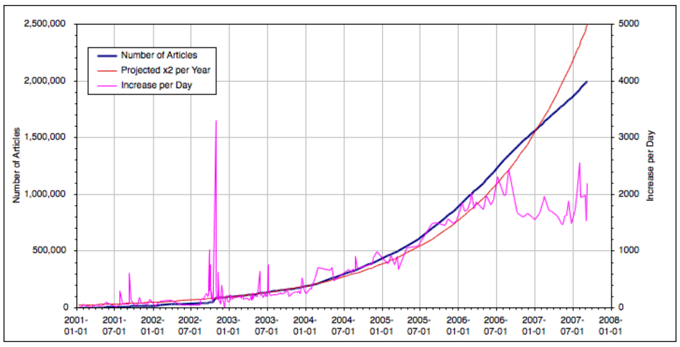


Figure 4.1: Article count for the English Wikipedia, from 2001 to 2007

A higher form of free divulgative activities are open learning initiatives, as promoted by several universities and colleges around the world; examples of

[2]http://www.wikipedia.org

[3]From http://en.wikipedia.org/wiki/File:EnglishWikipediaArticleCountGraph_linear.png

these open learning portals include MIT OpenCourseware [4], Stanford School Engineering Everywhere (SEE) [5], and Open Michigan [6].

Concerning open content, several sites propose free audio and graphics released under Creative Commons licenses: Jamendo [7], the Free Sound Project [8], Open Clipart Library [9], etc.

4.2 Open Tools and Applications

Free software has provided a major input in the creation of open content. The availability of completely open computing platforms (for example, GNU/Linux) has lowered the cost of ownership of computers with a free alternative to proprietary solutions (namely Microsoft Windows [10]). The beginning of free software can be dated back to 1984, with the creation of the GNU operating system and the publication of the first GPL license. The success of free software has been sustained by a large number of projects dedicated to both the creation of open alternatives [11] to proprietary applications as well as new and innovative solutions.

4.2.1 Free or Open?

Regarding software, very precise definitions of what is free and open source software have been made. According to the Free Software Definition [12], software is free if the end-user has the freedom to run, copy, distribute, study, change and improve it. Conversely, the Open Source Definition [13] provides a list of ten points that characterize open source software. It should be noted that the term Open Source was more of a marketing operation, started to make free software more appealing to industry. In particular, the main issue comes from the meaning of *free*, which refers to freedom (as *libre* in French) but can be intended as *free of cost* too (*gratis*). It is important to stress the fact that a major trait of free and open source software is the ability to distribute, change and improve existing software. This ensures that access to content created with such software is not dependent on a single entity as it is often the case with proprietary software.

4.2.2 Against software piracy

Computers, software and tools are of paramount importance in today's education programs. Not only do computers support the teaching and learning process, but for some classes they are essential tools. For some fields, the choice between proprietary and free platforms is hindered by the fact that no viable open solution exists. When free alternatives do exist, in many cases teachers

[4]`http://ocw.mit.edu/index.htm`
[5]`http://see.stanford.edu/SEE/Courses.aspx`
[6]`http://open.umich.edu/education`
[7]`http://www.jamendo.com`
[8]`http://www.freesound.org/`
[9]`http://http://www.openclipart.org/`
[10]`http://www.microsoft.com`
[11]`http://www.osalt.com/`
[12]`http://www.gnu.org/philosophy/free-sw.html`
[13]`http://www.opensource.org/docs/osd`

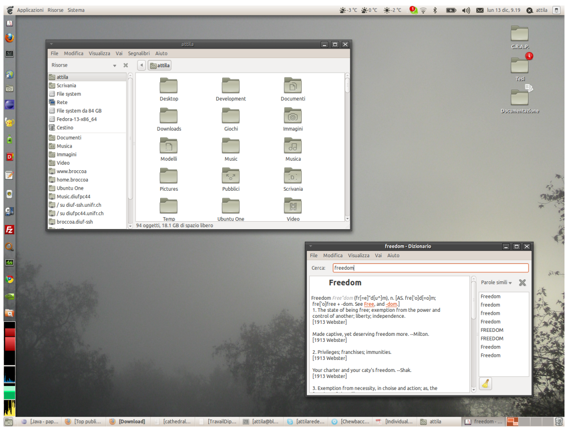
don't feel compelled to switch to free software, because they don't see any concrete advantage. Most users think that it is legitimate to copy and distribute proprietary software within educational institutions [26]. The need to educate teachers and students on the rights given by software vendors can be viewed as a way to increase awareness over illegal software distribution (also called *piracy*) [29]. In this context, free software represents an alternative that benefits both teachers and students: free software is easy to obtain, legitimate to copy, and enables both students and teacher to freely use the software at no cost.

4.2.3 Development tools

Although not necessary in the context of general education, in order to create open application and enable their modification according to the principles of free software, free development tools are required. In this context, free compilers and programming environments are essential to fully benefit from open source software. Most of the open source tools available today have been built thanks to the development tools created within the GNU Project [9]. For computer science students such tools represent an affordable alternative to expensive proprietary solution.

4.2.4 Operating systems

The first step in digitalizing education requires providing access to computers and their peripherals. Accordingly, operating systems need to be installed on computers. Because open source tools are typically available for a wide range of operating systems, users are not tied to a single platform, but are free to choose amongst proprietary or free operating systems. The quality and completeness of today's free operating systems (for example Ubuntu, shown in Figure 4.2), such as those based on the Linux kernel, rivals that of their proprietary counterparts (i.e. Windows and OSX). For educational tasks the investment required by free solutions is noticeably lower than that of proprietary ones, as no licenses are required to run software on multiple computers within the institution.



Freedom

Freedom Free"dom (fr[=e]"d[u^]m), n. [AS. fre['o]d[=o]m; fre['o]free + -dom. See Free, and -dom.]
1. The state of being free; exemption from the power and control of another; liberty; independence.
[1913 Webster]

Made captive, yet deserving freedom more. --Milton.
[1913 Webster]

2. Privileges; franchises; immunities.
[1913 Webster]

Your charter and your caty's freedom. --Shak.
[1913 Webster]

3. Exemption from necessity, in choise and action; as, the

Figure 4.2: GNOME desktop on Ubuntu GNU/Linux

4.2.5 Document editors

In the context of document editors, the most successful example is undoubtedly Openoffice.org [14]. The document writing and presenting capabilities of this free office suite represent an important contribution towards an open education, as both students and teachers are often required to write reports and dissertations or to present some subject.

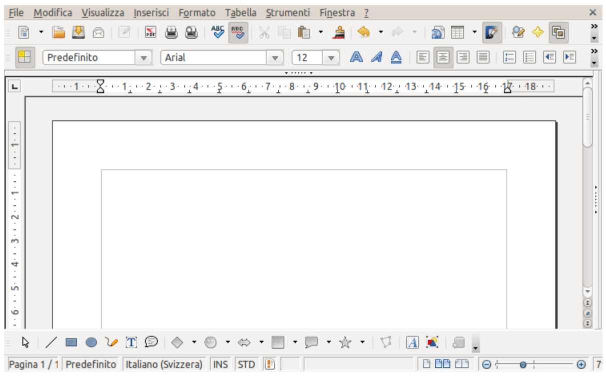

File Modifica Visualizza Inserisci Formato Tabella Strumenti Finestra ?
Predefinito
Arial
12
Pagina 1 / 1 Predefinito Italiano (Svizzera) INS STD

Figure 4.3: Openoffice.org Writer application

This office suite is also available on multiple operating systems, meaning that students and teachers are not tied to a single platform, and is translated into various languages. Moreover, as the software is available at no cost, students can freely use it at home for their homework (unlike proprietary solutions like Microsoft Office which would require a personal license even for home use).

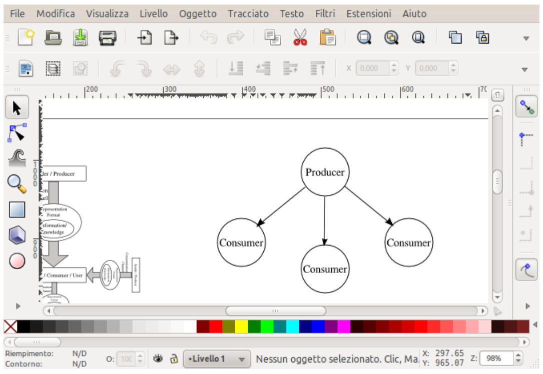
4.2.6 Graphic editors

Since the best way to spread information is not always textual there is a major need for tools allowing for creating and modifying figures, schemes and drawings. The open source community provides, amongst others, two tools that allow for handling graphical elements in a professional way: GIMP [15] and Inkscape [16] (shown in Figure 4.4). Besides being open source and free to use both are available for multiple operating systems (e.g. Linux, Windows, OSX).

GIMP (GNU Image Manipulation Program) allows for creating and manipulating graphic images. Its functionality encompasses photo retouching, image composition, and image authoring and is a valid alternative to commercial products such as Adobe's Photoshop and Illustrator.

In contrast to GIMP and other raster (bitmap) graphics editors Inkscape is an open-source vector graphics editor perfectly suited for web graphics, technical diagrams, icons, creative art, logos, etc. which is currently widely used by the community (for example, thousands of images on Wikipedia are created with Inkscape).

[14]http://www.openoffice.org
[15]http://www.gimp.org
[16]http://www.inkscape.org


File Modifica Visualizza Livello Oggetto Tracciato Testo Filtri Estensioni Aiuto
Producer
Consumer
Consumer
Consumer
Riempimento: N/D
Contorno: N/D
Livello 1

Nessun oggetto selezionato. Clic, Ma

Figure 4.4: Inkscape SVG Editor

4.2.7 Web tools

The World Wide Web is an inexhaustible source of data and information that can be visualized by means of web browsers. The open source community provides, since 2004, one of the most customizable web browsers of the world: Firefox [17] (Figure 4.5). Firefox was born from the Mozilla [18] project, which in turn was based on the source code of a proprietary web browser: Netscape Navigator. Thanks to the open source community Firefox continuously improves and thousands of add-ons have been created and are available for free.

File Modifica Visualizza Cronologia Segnalibri Strumenti Aiuto
http://www.mozilla.com/en-US/
Google
Mozilla | Firefox web browser & Thun...
mozilla
Products
Add-ons
Support
Community
About
Firefox 3.6
Download Firefox - Free
3.6.13 for Linux i686
Italian (9.4MB)
Completato

Meet the world's best browser, made just the way you like it. Learn more.

Figure 4.5: Mozilla's Firefox web browser

Alongside with Firefox, e-mail clients such as Thunderbird [19] help people communicate using free software. Furthermore, because of a strong link between open source communities and the Internet, a number of tools have been created for supporting different web related activities such as transferring files, text and

[17] `http://www.firefox.com`
[18] `http://www.mozilla.org`
[19] `http://www.mozillamessaging.com`

video conferencing, etc. Because communication is essential in learning, these tools can be successfully employed in e-learning scenarios.

4.2.8 Scientific Tools

Besides finding and producing scientific documents most of the time is spent by research in processing and analyzing data. For this purpose several open source software have been developed over the years for handling specific problems: In the following we review some of them.

Scilab [20] (Figure 4.6) is a free platform for numerical cumputation developed by INRIA (French National Institute for Research in Computer Science and Control) which is a worldwide reference software in academia and industry. Scilab's main features range from linear algebra, polynomials and rational functions to differential and non-differential optimization, while being mostly - but not completely - compatible with the reference commercial product of this domain, Matlab[21].

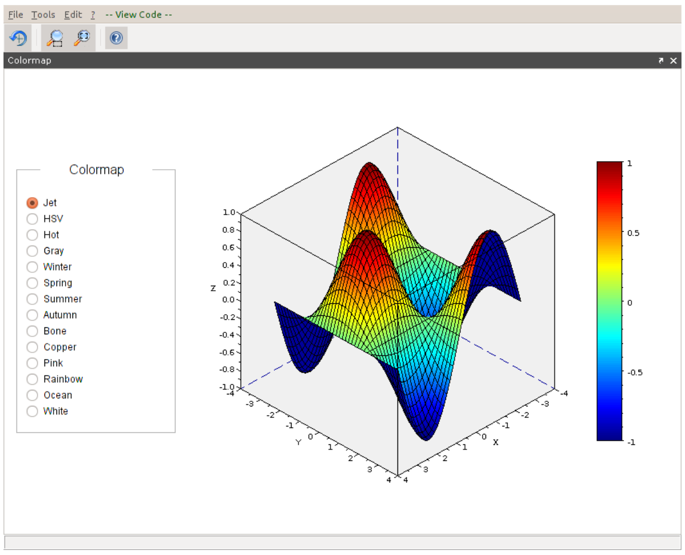


Figure 4.6: Scilab free platform for numerical computation

Similarly to Scilab, Octave[22] is primarily intended for numerical computations and provides means for solving linear and nonlinear problems numerically, and for performing other numerical experiments using a language that is mostly compatible with Matlab.

Another example of the success of open source software in the scientific domain is the R project[23], which has become a de facto standard among statisticians for the development of statistical software.

[20] http://www.scilab.org/
[21] http://www.mathworks.com/products/matlab/
[22] http://www.gnu.org/software/octave/
[23] http://www.r-project.org/

The common characteristic of the aforementioned software projects is their openness along with the fact that they are developed for several operating system, making thus them available to all kinds of users. Moreover, the open source nature of these projects offers a considerable advantage in terms of documentation and help along with several specific plug-ins made available by the community.

4.3 Open Formats

Beside open source applications and tools, essential to the development and spread of free knowledge in today's digital age, are open formats. As pointed out in [13], open standards are necessary to support interoperability and platform independence, maximize access, and provide long-term access to information. An open format relies on a publicly available description of the storage details, not encumbered by patents or any other form of restriction. The details of a format allow for the implementation of free editors and viewers, and ensure the survivability of the data in the future. A number of open formats exists and have been embraced by a number of software vendors (either open source or proprietary). In this section, the mainstream formats will be presented and reviewed.

4.3.1 XML

XML (eXtensible Markup Language) is a structured text format that is used as base for a large number of open formats (for example ODF and SVG). Because several tools exist to manipulate XML, applications can easily implement it. However, it is important to note that although the XML format itself is open, without a clear specification on how the information is structured within the document, it is difficult (sometimes almost impossible) to extract its contents.

4.3.2 Document formats

The Open Document format (also known as ODF), which stands for OASIS Open Document Format for Office Applications, is targeted at office data such as text documents, spreadsheets, graphs, and presentations. In contrast to proprietary formats such as Microsoft's Office ones, the full specification of the format is freely available and has been implemented by the OpenOffice.org suite. The format has been adopted by a number of governmental entities as a standard for public administration data. In 2007, ODF became an ISO standard, which proved its widespread support by major software developers such as Adobe, IBM and Sun Microsystems (now Oracle). In order to compete, Microsoft had to open their own formats [24] and release their specifications.

Another document format well known amongst the scientific community is LaTeX. LaTeX is a document markup language and document preparation system based on the typesetting system TeX [25]. Latex is used to format scientific documents such as technical reports, journal articles, and conference proceedings. Latex documents are text files written using a special syntax that describes

[24] http://en.wikipedia.org/wiki/Office_Open_XML
[25] http://en.wikipedia.org/wiki/TeX

the structure of the document rather than the details of the formatting. Documents can be transformed into *publishable* documents (for example in the PDF or Postscript format) by a compiler, which combines the Latex code with a template file that defines how the document is to be rendered. Being an open source project, Latex is available on a number of platforms, and enables high-quality document formatting and quick reformatting of the content.

4.3.3 Graphic formats

The Portable Network Graphics [26] format was created in 1995 as a replacement for the then patent encumbered format GIF [27]. As a fully documented open format, several tools are available to create, view, and manipulate PNG images. Because of its open nature, distribution of open source graphic content is typically achieved using this format. Another format worth citing is the SVG one [28]. While PNG is suitable for storing pictures, for vector graphics SVG enables scalable pixel perfect rendition of vector images.

4.3.4 Media formats

Similarly to PNG, the Vorbis audio codec [29] was created as an alternative to the patent encumbered MP3 [30] format, that required paying licensing fees for its usage. Vorbis encoded audio is typically distributed in the form of an OGG file (where OGG is the name of the container). Vorbis is not the only free audio codec, several others exist such as Speex [31] or FLAC [32]. Concerning videos, several open compression codecs are available, for example Theora [33] and WebM [34].

4.4 Open Web Content Managers

In addition to the ability to find, create and modify content by means of the aforementioned tools and formats, a further step is required to promote the sharing of knowledge to specific target groups. This is the aim of virtual learning environment (VLE), which are systems aiming at facilitating both the teaching and the learning process by providing tools specifically designed for this purpose. To avoid the ambiguity resulting from the plethora of terms that are used in different countries and communities to define similar environments, such as Learning Management Systems, Content Management Systems, Learning Content Management Systems, etc. we will follow the definition of VLE given in [6] which states that a VLE can be identified by the following features:

- A VLE is a designed information space

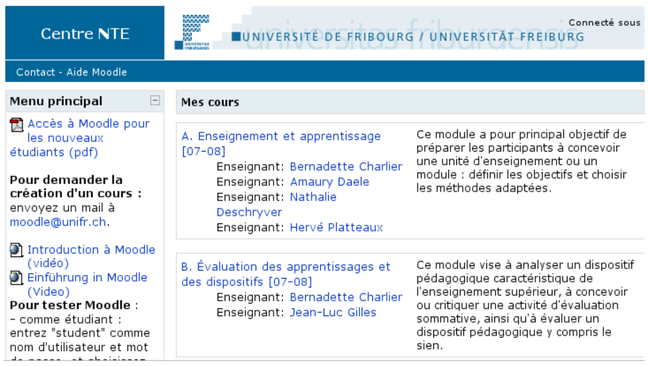
[26]`http://en.wikipedia.org/wiki/PNG`
[27]`http://en.wikipedia.org/wiki/Gif`
[28]`http://en.wikipedia.org/wiki/SVG`
[29]`http://www.vorbis.com/`
[30]`http://en.wikipedia.org/wiki/MP3`
[31]`http://www.speex.org`
[32]`http://flac.sourceforge.net/`
[33]`http://www.theora.org/doc/Theora.pdf`
[34]`http://www.webmproject.org/`

- A VLE is a social space (education interactions occurs in the environment)
- The virtual space is explicitly represented
- Students are not only active but also actors (they co-construct the virtual space)
- VLE are not restricted to distance education but they also enrich classroom activities
- VLE integrate heterogeneous technologies and multiple pedagogical approaches
- Usually VLE overlap with physical environments

Among the multitude of open source projects in this domain we choose to present here the most popular and, in our opinion, most interesting projects highlighting the benefits of their openness.

4.4.1 MOODLE

MOODLE is an acronym for Modular Object-Oriented Dynamic Learning Environment[35] and it is one of the most popular open source Course Management Systems with more than 37 mio users all around the world as of 2010[36], especially within universities (Figure 4.7 shows the MOODLE web site of the University of Fribourg). MOODLE aims at assisting the educators in the cre-



Accès à Moodle pour les nouveaux étudiants (pdf)

Pour demander la création d'un cours : envoyez un mail à moodle@unifr.ch.

Introduction à Moodle (vidéo)
Einführung in Moodle (Video)
Pour tester Moodle :
- comme étudiant : entrez "student" comme nom d'utilisateur et mot

A. Enseignement et apprentissage [07-08]
Enseignant: Bernadette Charlier
Enseignant: Amaury Daele
Enseignant: Nathalie Deschryver
Enseignant: Hervé Platteaux

Ce module a pour principal objectif de préparer les participants à concevoir une unité d'enseignement ou un module : définir les objectifs et choisir les méthodes adaptées.

B. Évaluation des apprentissages et des dispositifs [07-08]
Enseignant: Bernadette Charlier
Enseignant: Jean-Luc Gilles

Ce module vise à analyser un dispositif pédagogique caractéristique de l'enseignement supérieur, à concevoir ou critiquer une activité d'évaluation sommative, ainsi qu'à évaluer un dispositif pédagogique y compris le sien.

Figure 4.7: Moodle at the University of Fribourg

ation of course web sites by providing tools allowing to easily upload and share course materials, gather and review assignments while additionally providing chat and forums functionalities.

The open source nature of the project is certainly one of the major reasons for the popularity of MOODLE which thanks to its huge and engaged community offers more than 700 modules and plug-ins spurred from the different needs of its users.

[35]`www.moodle.org`
[36]`http://moodle.org/stats`

4.4.2 Claroline

Claroline [37] is a widely used open source eLearnig and eWorking platform which counts, as of 2008, more than 200000 installed versions on more than 100 countries, mainly in schools and universities (e.g. the University of Neuchâtel) but also in training centres, associations and companies. It comprises a list of tools allowing for publishing documents in several formats, prepare and manage online exercises, managing an agenda highlighting tasks and deadlines while also providing means of communication such as public and private forums.

The impact of the openness of Claroline is made clear by the several forks of the project that have given birth to new independent communities such as Dokeos[38] (in 2004) and Chamilo[39] (in 2010).

4.4.3 Wikis

A very interesting way to share and collaboratively manage information is provided by Wikis, which are websites allowing for easy creation and editing of interlinked web pages via a web browser. The use of a simplified markup language or a WYSIWYG (What You See Is What You Get) text editor make them particularly suited for non IT experts to work on collaborative knowledge.

The most famous example in this domain is certainly the MediaWiki[40] package which was originally developed for the free encyclopedia Wikipedia[41].

Along with MediaWiki, other wiki projects have been developed. One of the most interesting is DokuWiki[42], an open source wiki software which mainly aims at creating and managing documentation. While being similar to other wiki software, Dokuwiki is particularly easy to install and manage, because it doesn't require a database (all data is stored in plain files which remains readable also outside the wiki). Hence, Dokuwiki is particularly well suited for creating ready to go collaborative spaces where students can discuss ideas and keep track of them such as for projects of groups of students.

4.5 Open tools for open content for an open education

In this chapter we presented some of the existing open solutions that can be employed in an educational context. We have presented examples of free content available on the Internet, and listed some of the free tools that enable the creation of new content and modification of existing one. Furthermore, we have provided an overview of the formats that enable content to be shared and accessed by others and platforms that enable collaboration between people and support e-learning. In this regard, we argue that the technical limits to opening up education are minimal, and it is time for teachers and institution to really consider opening up the learning process.

[37] http://www.claroline.net
[38] http://www.dokeos.com
[39] http://www.chamilo.org
[40] http://www.mediawiki.org/wiki/MediaWiki
[41] http://www.wikipedia.org
[42] http://www.dokuwiki.org

Chapter 5

Open source in Computer Science classes

In the previous chapter we have dealt with a definition of openness, and listed several examples of successful open source platforms. In the following of this paper we will focus on the impact that open source has on teaching computer science at the university level. In this regard, two different concerns can be considered; on one hand, computer science students need to employ software in order to complete their exercises or to experiment with some technology. On the other hand, the goal of the university is also to teach how existing software works, and how to write new software. Whereas for the first concern the same examples as proposed in the previous chapter are valid, for the second concern a more in depth discussion is necessary.

It is our understanding that to better learn some study matter a complete and unobstructed view of each subject must be given. As a simple analogy, if somebody wants to learn the mechanics of a car, he should be able to work on a real car and explore its inner working: the same should happen to the teaching of computer science. Because open source software can be freely studied and modified, it represents an ideal field for experimenting with software for computer science students.

The rest of this paper thus explains the benefits of employing open source software and open learning techniques, analyzes some of the possible issues and purposes some solutions to mitigate such problems.

5.1 Teacher perspective

With respect to the previous analogy, being able to link the knowledge commonly found on textbooks with real-world examples empowers the teaching activity by providing a more complete overview of the subject. From our experience, while most of the traditional textbooks fail at providing specific documentation on how the presented topics are implemented in real systems, there exists a large technical literature (in most of the cases freely available) that covers almost all details about them. On the downside, constructing such link between theory and practice can be time consuming, especially when classes are based on dated textbooks for which it is difficult to find up-to-date concrete examples.

Another problem that is rooted in the very nature of open source is the ease of copying from existing solutions, thus the plagiarization of others' work. This problem can be overcome by introducing new challenges that require more than just wrapping existing solutions, for example by letting students work on real world problems such as in Google's Summer of Code initiative [1].

5.2 Student perspective

Students benefit from open source software and open learning techniques because of the additional amount of information that is made available to them. The ability to share information amongst students, retrieve existing information from multiple sources, achieve a better understanding of the topics being discussed in the class by digging into real world examples, represent an opportunity that should not be underestimated.

In this regard, the goals of Summer of Code initiative best resume the advantages of having students work on open source projects:

- Create and release open source code for the benefit of all
- Inspire young developers to begin participating in open source development
- Help open source projects identify and bring in new developers and project managers
- Provide students with the opportunity to do work related to their academic pursuits during the summer
- Give students more exposure to real-world software development scenarios

Because the source code in open source software is typically written *by programmers, for programmers*, the effort required by the student might be higher than with simple examples made for educational purposes. However, the availability of source code that can be reused within academic projects enables students to go further and stop reinventing the wheel.

5.3 Case Study: Operating System Project

In this section we present our experience and findings concerning the use of open source software in the context of a class teaching the principles of operating systems. The operating system course is a 3rd year (5th semester) mandatory class for students with computer science as their main branch. In the past three years (2007-2009), the authors have been involved in the exercise and project activities of the class.

On the theoretical side, the class was based on a well-known book by Prof. Tanenbaum [28], which provides an easy to read and understand overview of

[1] http://code.google.com/intl/it-IT/soc/

all the concepts surrounding operating systems. On the practical side, a programming exercise focusing on a real operating system kernel [2] (i.e. Linux) was proposed. In order to bridge the gap between the theory discussed in the class and the programming work, two additional books and presentation slides were employed.

From our point of view, we concentrate on describing the experience we had in managing the project. Having complete freedom on this part of the class enabled us to focus on the objectives that we wanted to attain. Concerning both the class and the exercise part, the direction was given by the textbook and by the professor (namely Prof. Béat Hirsbrunner and Dr. Michèle Courant).

5.3.1 Previous years

In the years before 2007 the operating system project was based on another operating system, named Minix [3], whose source code was also available. Unfortunately, the license terms for using such source code made it impossible to use it for non-educational purposes. While such restriction might seem of low importance for a computer science class, we felt that working on a project that could not be used outside the university was not motivating for students.

Moreover, for organizational needs, the original project would have needed to be reformulated in order to avoid plagiarism with previous years' solutions. Accordingly, we decided to completely turn the page and start a new project from scratch, by integrating novel ideas. It was our opinion that working on a widely used operating system such as Linux would raise students' interest and would represent a *plus* on their curriculum.

5.3.2 The class

The Operating Systems class (IN.5012) is targeted at 3rd year computer science bachelor's students (5th semester) and provides 5 ECTS. The class is also open to people in the business informatics branch, and consists in 2 hours of teaching followed by 1 hour of exercises. As a prerequisite, students must know the C programming language (1st semester class) and have followed the algorithm class (3rd semester). Beside exercises focusing on theoretical aspects of the topic reviewed in the class, a mandatory practical project is proposed. In each of the considered semesters, on an average of 13 students followed the class.

5.3.3 Project's overview

The goal of the project was to create a module (i.e. a software component) for the Linux [4] kernel that simulates a communication channel, technically named *pipe*. Such development requires the understanding of different key concepts of operating systems, such as locking mechanisms, process management, and device access. These topics are discussed theoretically in the class and are thus expected to be known by the student.

[2]The kernel is the part of the operating system responsible for running applications and enabling access to hardware devices

[3]http://www.minix.org

[4]http://www.kernel.org/

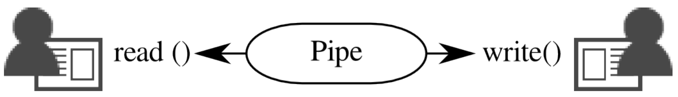


Figure 5.1: Pipe

Figure 5.1 shows the basic behavior of a pipe. An application can use a function `read` to read from the pipe, whereas another application can put data into the pipe by calling the function `write`. In this sense, two processes communicating through a pipe can be viewed as two persons speaking on the phone. A pipe implements a synchronous communication channel, meaning that certain semantic aspects need to be considered:

- only one application at time can access the pipe for writing;
- the pipe itself can store some information waiting for the reader to start reading it;
- when the space in the pipe is full, the writing application must be suspended (so as not to write anymore data);
- if the writing application has been suspended, it must be resumed as soon as there is space for writing in the pipe;
- when there is no information to be read, the reading application must be suspended;
- as soon as new data is available, the suspended reading application must be resumed.

Beside functional aspects, students were required to write a technical report of their module and to put comments in the source code (in order to make understanding easier). Furthermore, an oral presentation of the work done was scheduled at the end of the semester.

Developing for the Linux kernel does not require any proprietary software; as such, students had the opportunity to use free software only (compiler, editors, etc.). Finally, in order to avoid proposing the same goals every year, small changes were made to the project: more specifically, in the second year (2008) the pipe device was also required to perform some sort of encryption of the data, whereas in the third year (2009) the data written to the pipe would have made the keyboard light flash as in Morse code.

5.3.4 Didactic aspects

The proposed project had different didactic aspects and goals, which can be resumed as follows:

Know how to redo/know how to repeat Understanding of the fundamental concepts of kernel programming, modules, devices interfaces and process management. Understanding of the inner working of a pipe device, its semantic and the access functions (read, write).

Convergent know-how Programming of a Linux device module implementing a pipe. Implementation of a sample application which uses a pipe for communication.

Divergent know-how System and kernel development issues. Improving the knowledge of the C programming language. Knowledge of low-level debugging techniques. Improving oral and writing skills through the final presentation and the written report.

Know how to be/know how to become Understanding the differences between system and application programming. Known how to test a device module.

It should be noted that we have formally devised such aspects only *a-posteriori*, because the project had already started when such knowledge was learned in the *Did@ctic* classes; the intended objectives were nonetheless the same.

5.3.5 Evaluation

The final grade for the project was composed of 1/3 for the code, 1/3 for the documentation and 1/3 for the presentation. As for exercise series, the topics discussed during the project (including proposed short exercises or to-do) were part of the final writte course exam. For the final grade, the project accounted for 40%, the other 60% being determined by the written exam. This final evaluation was based on several factors, which included the implementation of the module, the documentation and the final group presentation. For the programming task, students were allowed to work in groups of maximum 3 people. Along with the kernel module, an application that enabled testing of the module was requested. Each student handed in his/her own documentation/report; meanwhile each group was asked to work on its own solution: students were informed that plagiarism would not have been tolerated. If plagiarism was detected, each participant of any group who plagiarized another's work would automatically fail the project part of the course (grade of 1). The final group presentation (20 minutes) had the goal of evaluating the student's overall comprehension of the topic.

For the implementation, the following criteria were applied:

- correctness: the code must have done what it was supposed to do (according to the requirements given to the students). A test program was to be written, to clearly show that the requirements are fulfilled. A bonus could have been considered for each additional (that is, not included in the requirements) functionality.
- structure: the code of the module needed to be divided, as needed, into functions and modules.
- style: the code needed to have been easy to read, well indented, well commented, and use clear, self-explanatory variable and function names.

For the documentation, the following criteria were applied:

- contents: the documentation must state and define the problem, describe the implementation, the relevant details, important terms, as well as the test scenarios that have been considered.
- structure: the documentation must have been well organized and structured (introduction, methodology, implementation, results, conclusion)
- style: the documentation must have been written with an appropriate style (index, headings, formatting,...)

No limit was given on the length of the documentation. Nonetheless, the recommended number of pages was set to 8. Finally, concerning the presentation, the guidelines given to the students asked for a brief introduction of the project, followed by a description of the issues and problems encountered. The presentation also included questions from the professor and the assistants.

5.3.6 Schedule

Instead of forcing students to commit to fixed weekly deadlines, only a final date for handing in the project (at the end of the term) was set. Each week, during the exercise hour, a short presentation introduced the details of the techniques to be used to implement the concepts seen in the class and to advance in the implementation of the module. Furthermore, the assistants were available for helping students advance with the development and quickly resolve possible issues.

The project was divided in 11 weekly parts, each consisting of:

- reading: reading chapters and sections of the proposed literature;
- browsing: quickly reading short sections in order to gain some insights about the inner working of the kernel;
- experimenting: experimenting with tools and applications on the system (for example to test the behavior of the module);
- programming: the actual development of the module;
- solving exercises: solving problems on paper in order to understand how to implement concepts seen in the class.

Only the final prototype along with a written report and a test program were due and were expected to be delivered and evaluated: there was no requirement to hand-in weekly solutions for the proposed exercises concerning the project. Those exercises were meant to help the student schedule the project, better understand the system, test his/her knowledge, and evaluate his/her progress. For help, the assistants were at the disposal of the student in order to solve issues as soon as possible.

5.3.7 Resources

The project required several sources of documentation, namely textbooks and web resources. More specifically the following books were employed:

Modern Operating Systems [28] by Andrew Tanenbaum is widely used around the world as textbook for teaching computer science. All important topics concerning the inner-workings of operating systems are reviewed: basic concepts, system calls, security, common issues, etc. While being a very good theoretical book, it lacks a concrete link with actual operating systems, thus making it difficult to fully understand how some of the concepts really work. This book was mandatory for the class.

Linux Kernel Development [17] by Robert Love focuses on the design and implementation of the Linux Kernel. A number of the topics discussed in Tanenbaum's book are also present in this book, but in this case a link with their actual implementation within Linux is present. This book was not mandatory neither for the class nor for the project. Nonetheless, several copies were made available to students.

Linux Devices Drivers [4] by Jonathan Corbet, Alessandro Rubini, and Greg Kroah-Hartman, is a guide targeted toward the implementation of modules for the Linux kernel. By means of a number of examples, all important concepts are discussed and explained. This book was deemed mandatory for the project: fortunately, all its contents are released under an open licence, thus freely available on the Internet.

Furthermore, students were advised to browse the available Linux source code [5], fully exploiting the possibilities given by the open source nature of the project.

5.3.8 Summary of the experience

All students have generally been able to successfully conclude the project and gained a positive mark. As the evaluation was done on different facets of the project, no single point-of-failure penalized less-skilled students.

Although no formal evaluation of the project was conducted, the feedback received from students (either on the written report or during the presentation) enabled us to conclude that they have been generally satisfied by the topic and by their work. Unfortunately it is difficult to really compare the rating of students across the years, because no information is available about the evaluation of the previous year's project based on Minix. Furthermore, changes in the classes given in the first two year and the presence of students from the business informatics branch (with less programming experience), sometimes affected the general level of understanding.

For the teacher This project has enabled a link between the theory presented in the class and a real-world examples of a working system. It takes clearly some time to prepare all the documents and slides, and to assist students in their work, but the overall result is, in our opinion, worth the effort.

For the students The project has been successful in motivating students to carry out their tasks. The project has been deemed long, but careful planning and division of the tasks amongst the members of the group enabled all students

[5]http://www.kernel.org/

to complete their project on time. Students also appreciated the fact that they had a chance to see how a real operating system works, and also had the possibility to work on a widely used system. For some of them, the experience working with Linux benefited their career prospects. Some of the comments found in the written report include:

> “Grâce au développement d’un module pour le noyau Linux, nous avons amélioré notre compréhension de la communication interprocessus.”

> “Mit diesem Projekt wurde erfolgreich ein Kernelmodul geschrieben (. . .) Dabei wurde viel über die Funktionsweise der Gerätetreiber in Linux gelernt. Dieses Projekt stellt nun auch eine Basis dar, einen komplexeren Gerätetreiber zu implementieren, welcher auch eine Hardware ansprechen könnte.”

> “Das Projekt ist ein guter Einstieg in die Kernprogrammierung, weil man mit vielen Grundkonzepten Bekanntschaft macht. Was das Projekt als Einsteigerprojekt besonders auszeichnet, ist dass man diese zum Teil doch etwas komplizierten Strukturen, für die Geräteoperationen kaum mehr braucht. Trotzdem bleibt die Aufgabe auch dann noch interessant.”

> “At least it was fun to implement a real kernel driver. We had to come over a lot of little and also some big caveats. (. . .) Personally I feel know well enough educated to start with my own projects on kernel drivers and have also already something in mind.”

> “Le développement d’un pilote fonctionnant comme une redirection au travers d’un pipe a permis de se familiariser au base de développement commune à tous les types de driver, tout en mettant en pratique certaines notions du cours de système d’exploitation.”

Beside positive comments, some students noted how time was the biggest constraint: due to it being a long project students are required to work regularly and thus avoid rushing to finish the project in the last weeks. Conversely, real difficulties and frustration were encountered by some students in the business informatics branch, as it’s evident in some of the reports’ comments:

> “Grundsätzlich war die Implementierung eines Linux Kernelmoduls sehr interessant, schliesslich studiere ich nicht Informatik, wenn mich solche Dinge nicht interessieren würden.”

> “Nun es mag ja gut sein wenn ein angehender Informatiker schon während des Studiums mit der Implementation eines einfachen Kernelmoduls konfrontiert wird. Andererseits ist es fraglich ob es wirklich Sinn macht die Studenten schon im Grundstudium mit einem solchen, doch sehr spezifischen, Problem zu beschäftigen. Ich bezweifle sehr dass wirklich jeder Informatiker während seines Berufslebens mal einen Linuxkernel implementieren muss.”

To reply to some of the comments from students in the business informatics branch were that negative toward the project, we should note that the class was mainly addressed towards students with computer science as main branch. Accordingly, the requirements were set a little bit higher than what is probably expected in business informatics classes.

5.4 Closing remarks

Our experience with this project has been generally positive. The possibility of working with open source technology has not only benefited students' learning but has also been pushing us forward and through our engagement in such projects we too have gained some experience. Nonetheless, there are several points that could be changed in order to improve the project. First, the didactic aspects and pedagogical goals where never clearly defined nor discussed with the students. In fact, students were only aware of the practical *goals of the project*, namely the development of a Linux kernel module. We think that if the didactic aspect of the project is put forward and exposed to the students, their motivation can increase. Furthermore, the project left little space for students' creativity: the task was clearly assigned and the time to introduce personal modifications was scarce. In this context, it should be noted that it is not feasible to leave complete freedom to the student, as the effort required for assisting him would be too much.

Chapter 6

Conclusion

The open source philosophy has changed the way content is created and shared among people. Starting with free software, people have begun to share their work for the benefit of others, and have enjoyed seeing their products being embraced and improved by others. The same ideas have spread in other fields, such as graphic arts, knowledge, or music and have resulted in a large amount of open content that can be used, re-used, and re-distributed under permissive licensing terms.

In this paper we presented an overview of open source and open content usage in education. We first defined what open source and open content are, and highlighted the differences between the terms copyright, ownership and licenses. Moreover, we listed a number of examples of open source content and software that are used today by millions of people.

Concerning the use of open source in computer science classes, we discussed the related benefits and the issues. Finally we presented our teaching experience within an operating system project.

Unfortunately, our experience at the University of Fribourg has ended, meaning that our effort to improve the Operating Systems class cannot go further. We had hoped that our colleague who has taken over the supervision of the project would continue on our path, but unfortunately the project has been scrapped for lack of resources. We are nonetheless confident that it could be resumed in the future.

The *take home message* for this paper is that open source can have a positive impact on teaching computer science. Students get motivated to work on existing open source projects and feel that this experience can improve their general knowledge of a topic.

Bibliography

[1] Magnus Bergquist and Jan Ljungberg. The power of gifts: organizing social relationships in open source communities. *Information Systems Journal*, 11(4):305–320, 2001.

[2] Dana Blankenhorn. Obama enforces trade embargo against open source, Last Accessed December 2010. `http://www.zdnet.com/blog/open-source/obama-enforces-trade-embargo-against-open-source/5698`.

[3] C. Coppola and E. Neelley. Open source open learning: Why open source makes sense for education. 2004.

[4] Jonathan Corbet, Alessandro Rubini, and Greg Kroah-Hartman. *Linux Device Drivers, 3rd Edition.* O'Reilly Media, Inc., 2005.

[5] The MITRE Corporation. Use of free and open-source software (foss) in the u.s. department of defense, 2003.

[6] P. Dillenbourg, D.K. Schneider, and P. Synteta. Virtual learning environments. *Proceedings of the 3rd Hellenic Conference "Information and Communication Technologies in Education"*, pages 3–18, 2002.

[7] Cory Doctorow. "intellectual property" is a silly euphemism, Last Accessed December 2010. `http://www.guardian.co.uk/technology/2008/feb/21/intellectual.property`.

[8] Electronic Frontier Foundation. Open audio license, Last Accessed October 2010. `http://en.wikipedia.org/wiki/Open_Audio_License`.

[9] Free Software Foundation. Gnu, Last Accessed October 2010. `http://www.gnu.org/`.

[10] Free Software Foundation. Gnu free documentation license, Last Accessed October 2010. `http://en.wikipedia.org/wiki/GNU_Free_Documentation_License`.

[11] Free Software Foundation. Gnu general public license, Last Accessed October 2010. `http://www.gnu.org/licenses/gpl.html`.

[12] Harald Welte, Last Accessed October 2010. `http://gpl-violations.org/`.

[13] Brian Kelly, Scott Wilson, and Randy Metcalfe. Openness in higher education: Open source, open standards, open access. In *ELPUB*, pages 161–174, 2007.

[14] Alex Koohang, Eli Cohen, and Keith Harman. Open source: A metaphor for e-learning. *Informing Science Journal*, 8, 2005.

[15] Shaheen E. Lakhan and Kavita Jhunjhunwala. Open source software in education. *Educause Quarterly*, (2), 2008.

[16] Lawrence Lessig. Creative commons, Last Accessed October 2010. `http://creativecommons.org/`.

[17] Robert Love. *Linux Kernel Development (2nd Edition) (Novell Press)*. Novell Press, 2 edition, January 2005.

[18] Wolfgang Maass. Inside an Open Source Software Community: Empirical Analysis on Individual and Group Level. pages 66 – 71.

[19] Massachusetts Institute of Technology. Mit license, Last Accessed October 2010. `http://en.wikipedia.org/wiki/MIT_License`.

[20] Open Knowledge Foundation. Open definition, Last Accessed September 2010. `http://www.opendefinition.org`.

[21] Open Source Initiative. Open source definition, Last Accessed September 2010. `http://www.opensource.org`.

[22] Young Jeffrey R. Five challenges for open source. *Chronicle of Higher Education*, 2004.

[23] Eric S. Raymond. *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2001.

[24] Regents of the University of California. Bsd license, Last Accessed October 2010. `http://en.wikipedia.org/wiki/BSD_licenses`.

[25] Oshani Seneviratne, Lalana Kagal, Daniel Weitzner, Hal Abelson, Tim Berners-Lee, and Nigel Shadbolt. Detecting creative commons license violations on images on the world wide web. In *WWW2009*, April 2009.

[26] Robert M. Siegfried. Student attitudes on software piracy and related issues of computer ethics. *Ethics and Inf. Technol.*, 6:215–222, December 2004.

[27] Richard Stallman. Did you say intellectual property? it's a seductive mirage, Last Accessed December 2010. `http://www.gnu.org/philosophy/not-ipr.xhtml`.

[28] Andrew S. Tanenbaum. *Modern Operating Systems*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2007.

[29] George Teston. Software piracy among technology education students: Investigating property rights in a culture of innovation, 2008.

[30] Sam Williams. *Free as in Freedom: Richard Stallman's Crusade for Free Software*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2002.