Semana 1
Software Libre

Lectura
Introduction to Free Software.

Barahona, J. G. (2009). Introduction to Free Software. Barcelona: Eureca Media, SL. ***Unidad 1****. Introduction (pp. 9-11)* ***Unidad 5****. Economy sección* ***5.1*** *Funding free software projects (pp. 66-73) y* ***5.2*** *Bussines models based on free software (pp. 73-79)*

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.

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 gnuLinEx, 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. 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 (2001) [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 it is not easy to use those based on the sale of licences, the most common found models in software industry, 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 his or her 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, 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: