Advice report

EUROPEAN COMMISSION

Enterprise Directorate General

IDA/GPOSS

Encouraging Good Practice in the use of Open Source Software in Public Administrations

GUIDELINE FOR PUBLIC ADMINISTRATIONS ON PARTNERING WITH FREE SOFTWARE DEVELOPERS

Revised Final version: December 2005

Prepared by:

UNISYS

Rishab Aiyer Ghosh (Merit)

Ruediger Glott (Merit)
Gregorio Robles (Universidad Rey Juan Carlos)
Patrice-Emmanuel Schmitz (Unisys)

Disclaimer

1) Executive summary

In recent years, Free[1] / Libre / Open Source Software (FLOSS) has developed as a novel form of collaborative production. Since its origin as a collaboration between individual volunteers, it has seen tremendous success, both in terms of the commercial and technical strengths of the produced software itself, but also as a model of organisation and development. In particular, it has received much attention from public administrations (PAs) for two reasons: the software itself may be cheaper to use and support than proprietary software applications; and free software may be a novel, cost effective and highly responsive way to develop applications specific to PA needs. The second point takes advantage of the modifiable nature of free software, which makes it suitable for adaptation to PA needs.

PAs may be interested in working with free software projects in order to take advantage of their adaptability, low cost and the ability to engage with the large developer community. This document aims to help PAs achieve this successfully. It starts with an overview of the free software phenomenon, the structure of free software project organisation and developer motivations. Following this is an overview of how free software projects respond to external inputs, financial as well as on user needs. Chapter 4 provides step-wise guidelines on how PA's can successfully engage with free software communities, from the technical, social and economic perspectives.

Chapter 5 elaborates practical guidelines on the legal framework of collaboration between free software developers and PA's: solution will greatly differ depending on the fact the software was developed by the PA, for the PA, or found externally to the PA; it will differ if the software answers to very generic or very specific needs: it will differ if the software was written from scratch or build from various components or building blocks found elsewhere with their own inherited licenses.

[1] „Free“ is always used here in the sense of freedom, except when we write „for free“.

It must be noted that successfully engaging with the free software developer community requires a good understanding of how and why free software development works, in order to determine why free software developers would want to collaborate with your specific PA-supported effort. To support this, Appendix 1 provides more details on the workings and motivation of community participants and Appendix 2 provides details of how some successful free software projects actually work.

2) Introduction to free software development

a) What is free software?

In recent years, Free / Libre / Open Source Software (FLOSS) has developed as a novel form of collaborative production. It started as a collaboration between individual volunteers, often academics or students. It has grown into something that the world's biggest companies and public organisations depend on as users and as businesses[2]. It has seen tremendous success, both in terms of the commercial[3] and technical[4] strengths of the produced software itself, but also as a model of organisation and development: free software is arguably one of the best examples of open, collaborative, internationally distributed production and development that exists today. This international aspect has resulted in tremendous interest in the domain of free software from around the world, from all communities: government, policy, business, academic research.

Although Free/Libre/Open Source Software is not a new phenomenon, it has considerably increased in importance in recent years. It is important to distinguish between the software itself, and the organisation of its production. The terms "free software" and "open software", while defined differently (the original four freedoms of the Free Software Foundation and the much later nine conditions of the open source software definition[5]) both refer, strictly speaking, to software itself, and not the process of its creation. In particular, they refer to software that is released under certain licences,

---

[2] Companies like IBM and Oracle have announced billions of dollars of marketing spending for free software-related businesses. A study for the US Department of Defense found that stopping their use of open source would have "immediate, broad, and strongly negative impacts… on the ability to defend against cyberattacks" (MITRE 2003); websites from Google and Amazon depend on free software in order to run. Even the World Intellectual Property Organisation, ironically, uses Apache and Linux on its web server.

[3] E.g. HP reports $2.5 billion in Linux-related revenues, growing at 40% annually: http://www.infoworld.com/article/04/01/15/HNhplinuxrevenue_1.html

[4] Performance and security are the top reasons for organisations to choose free software according to the EC-financed FLOSS User survey: http://flossproject.org/report/index.htm

[5] http://www.fsf.org/philosophy/free-sw.html and http://www.opensource.org/docs/definition.php

approved by the Free Software Foundation or by the Open source Initiative (OSI). As it turns out, the list of approved licences is nearly identical.

It is entirely possible for free software to be written within a company – or public administration – exactly as if it were intended for commercial release as proprietary software. If this software is distributed under a free software / free software approved licence, it is free software. The purpose of this guide, however, is to help public administrations tap into the enormous resources promised by the most commonly associated form of production for such software – a large, distributed volunteer community of developers collaborating in order to produce and improve a software product. The software licence facilitates such collaboration, as it ensures that contributors are not hindered in their attempts to improve a given software product – the licence allows anyone the freedom to use, copy, study, modify and redistribute it, without which spontaneous or voluntary contributions would be impossible.

The software licence certainly does not guarantee such collaboration. Releasing a first version of a citizens land records management system under a free software licence will allow the free software community to modify and improve the system. But it certainly does not guarantee this. In order to improve the ability to work with the community of contributors, it is important to understand the collaborative software development process.

Many aspects of this process still appear unknown or even strange. Economic exchange relations, as they occur within the community of free software developers as well as in the traditional parts of capitalist economies, are usually based on the fundamental principles of monetary payments for production and transactions. However, these principles seem not to apply to much of the productive activity around free software (although money is very much present in the related activity of servicing free software systems), and still this domains functions very well and gains more and more importance in the leading software markets.

This section provides a brief overview of the organisation, leadership structure and motivations of free software developers. A more expanded view is found in Appendix 1, and several resources are available for further details. It is important to realise that the success of partnering with free software developers for public administrations (or anyone else) depends on a good understanding of how and why free software communities work. PAs who simply publish requests for support for the creation of new software, or release some software onto the Internet, and then expect a vast community of volunteers to selflessly attend to their needs are likely to be disappointed.

b) Organisation of free software developer communities

The development of software and the cooperation in free software developer communities differs considerably from developing proprietary software in firms. Free software is developed as a result of free cooperation of autonomous developers connected through a highly volatile network organization; proprietary software is usually a result of hierarchical teams working on clearly sequenced tasks. However, free software, like proprietary software, requires developers who write code, coordinate work, and maintain and administer the progress of a project. When a free software project reaches a critical size, individual people take response for different project tasks and coordinating institutions and governing structures emerge. These differ from project to project based on various criteria, including the project's ties to firms, size, and technical nature, and can include "benevolent dictators" (e.g. Linus Torvalds as leader of the Linux kernel development), "rotating dictatorship" (e.g. the Perl scripting language) and democratically elected committees (e.g. FreeBSD, Apache, Debian). Technical merits, effort contributed and reputation play a significant role in the determination of leaders and leadership structures. Based on studies of demographics it appears that free software developer communities contain not only technical expertise but also a high degree of management skills.

c) Motives of free software developers

There is a large variety of assumptions as to why people join the FLOSS community and "work for free". The first thing to realise is that developers, by and large, do *not* "work for free". They perceive themselves as selfish and self-interested, and expect returns from their contribution. These returns are not always monetary: learning new skills is the most commonly cited reason for developer participation. But monetary returns are an important benefit – roughly 50% of developers earn an income through their free software work, and this accounts for a large majority of the most productive and experienced developers.

Recognition by peers and society is a motivating factor, but the nature of reputation is such that it can reward a few leaders, but not the majority of voluntary participants. It is, however, a factor that can be artificially boosted by PAs at relatively low cost, by awarding prizes or other forms of public recognition to developers.

Another class of motivators is product-related: developers who have a need that an existing free software product can't meet may be willing to fix the software and contribute their fix back to the community. Similarly, the free software community is excellent at spotting and fixing bugs in their software.

One key underlying motivator, that often ties into those outlined above, is an overlap between individuals as developers and the same individuals as users. Generally speaking, the pool of developers who may find some self-interested motivation to contribute to a software project is proportional in size to the project's potential user base. This can be disadvantageous for PAs when they wish to build PA-specific software (and advantageous when they want generic software tailored to their needs, as with Extremadura's GNU/LinEx based on the generic Debian package). However, the free software ecology has many examples of niche communities of users with their own niche community of developers, from precision engineering to hospital information systems. This attribute can benefit PAs; it could lead to many developers from many PAs and PA-related businesses collaborating in a PA-specialised niche of developers.

PAs need to understand in depth what motivates developers, and before attempting to create community-based free software applications will need to have a clear understanding of why developers would be interested in collaborating with them. Appendix 1 provides some more details on developer motivations.

3) External incentives

a) How free software projects relate to external funding

As described above, free software developers – often through free software businesses that employ them – are encouraged through direct financial incentives. Meanwhile, users of free software are often satisfied enough to pay. Hiring an entire team of developers is not often an option for public administrations, however. Hiring some developers to lead a project (or contributing existing staff to the development process) may often be a good idea, as it provides an extent of control over the resulting software at least in terms of the technical features and timeliness of production. Contracting developers for service and support of related systems (particularly individual developers or SMEs) similarly is a direct and straightforward financial incentive, and has been used for example in the Spanish region of Extremadura (see section 3).

A more innovative approach is the “bounty” model, where an informal competition or fellowship scheme is run, to provide monetary or other material rewards to developers who solve particular problems. This has been pioneered by the software billionaire and philanthropist Mark Shuttleworth[6] in South Africa. In 2003, he had announced a bounty for development on SchoolTool, a free software school administration system developed in Python. After some proposals and development efforts, the core development team was selected – led by developer Steve Alexander, with members in Lithuania and the UK. The bounty was EUR 37,200 for work commencing 1 September 2003. Shuttleworth’s website announces further bounties for individuals or teams interested in extending SchoolTool to manage pupil academic progress and test results, willing to collaborate full-time with Steve Alexander's team.

The Indian branch of Red Hat announced “the first ever programme of its kind in the world[7]” – a scholarship programme somewhat similar to the bounty scheme, that provides funds for individuals or small groups, particularly students, who develop high

[6] http://www.markshuttleworth.com/bounty.html

[7] http://www.in.redhat.com/community/rhscholarship.php

quality free software solutions or contribute to free software projects. In this case, the funds are provided after the contribution is made.

b) How free software projects relate with specific user communities

A good example is in healthcare. The US Veterans Administration provides lifelong healthcare for military personnel, and the US Government developed and maintained a Healthcare Information System (HIS) called Veterans Health Information Systems and Technology Architecture (VistA). The huge software system used for this was proving expensive to maintain, and, under the US Freedom of Information Act, source code was released into the public domain. This resulted in an initial burden as users (other health systems and hospitals) assumed that the original authors would continue to maintain the system, while in fact the original authors wished to encourage a community to do so. Eventually, the VistA system is supported by foundations, businesses and the health community around the world, and is widely believed to be the largest and most widely used HIS application. While the software is free, it has historically run on proprietary systems (such as Windows or VMS) which support the specialised programming environment in which VistA is written. OpenVistA is a free software suite including VistA and enabling tools that allow VistA to run on GNU/Linux.

When the Beaumont Hospital, the largest public hospital in Dublin, decided to switch to free software (Fitzgerald and Kenny, 2003) it chose VistA as its hospital management system. While it got support from many members of the VistA community, including from hospitals elsewhere in Europe, it also joined the community as a contributor by providing its own feedback. It joined a large community of users that include the US military health system and the German Heart Institute, Berlin.

As this example shows, though, there is no single monolithic "Free software community" but lots of different overlapping communities and sub-communities addressing needs of different groups. These communities tend to automatically be close to the users whose needs they address as they often are formed out of those users, or subsets of the more technically adept users. So, free software in healthcare is largely supported by IT staff at hospitals and healthcare providers along with businesses and supporting organisations around them; free software for bio-tech (e.g. BioPerl, the basis for the Human Genome Project) originated at the European institutions contributing to the HGP. Similarly, free software for public administrations is likely to be supported by a community initiated by PAs themselves.

The next section looks in more detail at some major free software projects (not specifically related to PAs, but providing an essential understanding of how such projects function and therefore the context in which PAs must learn to collaborate).

4) How to collaborate with free software developers

The previous sections provided an introduction to the way free software developers function, how they organise into communities and how such communities address user needs and grow to absorb their users including those with special interests. Following case studies to provide an understanding of the context in which PAs must operate, this penultimate section outlines a step-wise approach that can improve the chances of fostering a successful collaboration between public administrations and free software developers.

a) Identify the “seed” – what will attract developers?

Free software is not developed in a vacuum. It requires an initial impetus, which is usually in the form of software itself. Some software is released as an attempted solution to a need, and forms the kernel around which more software is written and around which a community of developers forms. A general public call – from a PA or anyone – for support to solve a problem is unlikely to work unless accompanied by a demonstration of efforts made to solve it already. So the best way for a PA to engage a community is to start with the core of an application being developed in-house, or by subcontractors, and then released. This core application provides the seed around which a community of developers can form.

It is also possible to achieve a successful community collaboration if a group of PAs collaborate formally and develop the core of a software application that can be released as the “seed” that will grow into a supportive community. Either way, such a software seed is usually an essential pre-requisite to the self-sustaining process of stimulating the growth of a community.

b) Disseminate to reach out to the developer community

If nobody knows of your needs, nobody will respond to them. If you develop “seed” software but potentially interested users and developers (including other PAs) do not learn quickly about it, a community cannot form. Therefore dissemination that is widespread but reaches the target communities is essential. Several channels exist for such dissemination, but they are usually either for PAs (e.g. the IDA Open Source Observatory) or for developers (e.g. Sourceforge). Few dissemination channels are (currently) specifically addressed to PA developers, however. In some regions such channels may exist to some degree – e.g. through the AFUL[8] organisation in France, or through the extensive initiatives in Extremadura in Spain. More work is needed to develop dissemination channels for PAs interested in attracting developers and collaborators. Initiatives to develop such dissemination channels can involve interacting with local developer organisations where they exist, or engaging the community, e.g. Linux User Groups (LUGs) through PA participation (informal using individual PA staff, or formal through sponsoring events). Other such dissemination channels could include infrastructure platforms for developers (such as BerliOS, see point (e) below). The absence of well known dissemination channels for communication between PAs and developers is a problem yet to be solved.

c) Attract a community to solve a problem (e.g. LinEx, Spanish Linux for Extremadura)

This is an obvious goal. However, attracting a community to solve a problem requires that the previous steps are taken first – PAs should first show that they are taking the initiative to solving the problem themselves, e.g. by creating “seed” software, and that they are engaging with the community. If the PA efforts build upon applications with

[8] Association Francophone des Utilisateurs de Linux et des Logiciels Libres (Francophone Linux and Libre Software User Association), www.aful.org

existing developer communities, attracting this community becomes easier.. To take a concrete example, GNU/LinEx in Extremadura was developed as a localised version of the existing, widely supported Debian distribution of the GNU/Linux operating system. The choice of this distribution was important, as Debian is the only major distribution of this operating system for which development and integration is not led by an individual company (such as Red Hat, Mandrake or SuSE/Novell, which control the eponymous other major distributions). Debian is a purely community effort with hundreds of integrators responsible for configuring and packaging a suite of software applications developed by thousands of individual contributors. Thus Extremadura's first choice tied it to a community rather than an individual company, immediately embedding it within this community (Debian has an increasing number of developers from Extremadura and more generally Spain).

Debian was the foundation, and the seed of the GNU/LinEx project was a localised distribution paid for by the government. It hired a local software company to develop a localised distribution of Debian and released that into the wider community. This quickly developed into a growing local developer community with close links to the wider group of global Debian developers. It may not always, or even often, be possible to design the solution to a new PA problem around an application such as Debian, with a large, existing and very active community. However, PAs should carefully consider how a community could form around their own problem, and whether a community already exists that could be persuaded to pay attention to PA needs. As described in Chapter 2, on the external incentives of open source communities, active participation in getting major projects to work more in line with needs of PAs can work when PAs are a big user base and the solutions are not extremely specialised. This is, for instance, quite common in free software projects like MMBase (especially in the Netherlands where a number of PAs use it and a number of Dutch businesses have been built around it) or Zope. For more specialised needs, however, the strategy must be more along the lines of attracting a community of interested parties around the PA's specific needs, with the PA clearly showing (as in steps a and b outlined above, or through financial incentives described in Chapter 2) its own strong interest and initiative.

d) (or) Attract a community to support a pre-existing software application

Another alternative way of attracting a community is to release a fully developed application of wider interest – planting a tree rather than a seed. This is only possible when such a fully developed application is available and controlled by the PA. In such cases, this can often be more successful as the community is not invited to solve a problem, but to build upon an existing, working solution. Typically, this is software developed and maintained by public administrators – e.g. VistA, the hospital management solution – who wish to release it as free software for various reasons. These could be ideological, or practical – the PA may no longer have a budget to continuously maintain the software, and as the development costs are sunk costs, the PA may believe that the software would find a wider community of users who would share in the costs of maintenance following the free software model. In the example of VistA, a fully functional hospital management system was released as public domain (as all intellectual works developed by the Federal Government automatically are, under US law). The large global community of users that developed around this fully functioning solution led to a similarly large community of developers to provide support, maintenance and further development. While this model does not have the potential of the previous route (c) of saving initial development costs, it does allow sunk development costs to be leveraged when funds for further development, support and maintenance are limited or unavailable. In such circumstances, turning to a wider developer community through the application's release as free software may be a useful way for PAs to save further costs and derive the benefits of sharing responsibility for the software's further evolution.

e) Provide or identify collaborative development infrastructure (e.g. Berlios, US-based SourceForge)

PAs can provide themselves with channels to reach developer communities by creating development infrastructure. The American Sourceforge is the best known site that provides such infrastructure – publicity, webspace, technical, team-management and version-control utilities that can be indispensable for developers. Such infrastructure provides a hub around which developer communities form, and developers on one project in a given infrastructure portal are likely to notice and contribute to new projects on the same portal. Providing infrastructure is also a way of giving back to a community in exchange for their contributions to solving particular needs. An European example of a government supported infrastructure site is Berlios (Berlin Open Source), a Sourceforge like system that has grown into one of the biggest hosts of many German projects. While there is some discussion as to whether Berlios has been successful at attracting developers to projects initiated by PAs, it is clear that Berlios has become a major resource base for European and particularly German developers, with the potential for exploitation by PAs. This is in contrast to some other PA-related developer portals such as US-based Governmentforge.org or the global and rather more successful Schoolforge.org. PA-related software and developers interested in working on PA-related software can be found on these sites, but unlike Berlios, these sites did not arise as a result of PA initiatives. Rather, they were created by developers themselves, often with the aid of private-sector organisations with commercial interests in supporting the resulting software.

It is worth examining an infrastructure portal that is supported by a coalition of PAs, which could attract projects of interest to PAs and developers interested in involvement in such projects. Such a portal, "seeded" with PA-related software projects already developed by participating PAs, could be an attractive proposition for engaging developers in further developing such software.

f) Cooperate and proactively provide feedback

It should be emphasised that – as the Linux author Linus Torvalds said – users are important, because they form the basis for development in the free software model; therefore it is important to ensure that users in PAs interact extensively with developers whether such developers are in other PAs or elsewhere. This may seem obvious, but one of the main reasons for failure when PAs work with free software is a lack of interaction between users, developers and IT staff/managers. This is particularly true for migration efforts (see, e.g. IDA OSO case study on the German Monopolies Commission[9]), and

[9] http://europa.eu.int/ida/en/document/3277/470

similar problems due to poor user interaction will arise in the development or take-up of new free software applications. Feedback for free software development is considerably more important than for proprietary software for at least two reasons: free software developers are often more willing and able to act quickly on user feedback; and free software developers often work on a “user-pull” rather than “user-push” model, responding to user requests rather than forcing features upon them. In short, free software developers need feedback, and they usually act upon it quickly.

g) Identify community leaders

It is helpful to identify potential community leaders. While free software communities are self-organising, leaders are essentially self-selected in that those who show the most initiative become de facto leaders. It is useful to try to recognise such individuals and interact extensively with them.

As described in the appendix (“Organisation of free software communities”) leadership roles in free software projects are determined by the project’s organisational structure and these vary enormously. Leadership may be claimed by someone who writes the most software; someone who coordinates and makes decisions; someone who publicises the work of the community or encourages new developers. The importance of such roles depends on the nature of the project. For PA projects, the importance of leadership roles will depend at least on the degree of control the PA wishes to have and the degree of material support the PA provides. When material support is weak, leadership may tend towards individual contributors who write more software – i.e. are most productive. For such contributors, reputation can be an incentive, and recognition from PAs can help with this. Such engagement and recognition may also help PAs retain some influence over the project’s direction.

When material support is relatively strong, the PA may be responsible for financing much of the software development either through internal staff, subcontracting or through other financing models such as “bounties” (see chapter 2, “How free software projects relate to external funding”). In such situations the leadership sought from community participants may be more in terms of developers who promote the project within the community and lead to increasing community support. Of course, high visibility should be given to the developers who are actually receiving PA funding and thus represent the PA’s control over the project’s direction.

h) Identify selection mechanism to balance competition with cooperation between developers

The structure of free software development, while portrayed as cooperative and communal, is in fact a form of Darwinian competition at levels unheard of in proprietary software development (which has a top-down approach). In the free software model anyone is free to contribute; problems may receive multiple solutions, and the best technical solution is usually adopted. This competition is usually self-organising based on the expression of user needs as well as technical review by developer peers. But in case of specialised users such as PAs, it may be helpful for PAs to clearly express preferences for technical choices as and when they occur, and ensure the continuous monitoring of the technical choices made in order to have their preferences heard. While this reinforces the “user pull” described in point (f), it may help for PAs to express technical preferences also as a co-developer, through the individual voices of the leaders cultivated by the PA as described in point (g). Individuals are better placed in the dialogue of free software developers to clearly present a technical viewpoint, and numerous commercial organisations (such as IBM in Linux or Apache, Novell and Ximian with GNOME) have successfully “steered” community contributions to free software projects by getting their views heard through individual leaders in the community, often, but not necessarily, employed by them.

i) Identify funding methods if required

It is important not to ignore the possibility that funding methods may be required. These are likely to be inversely proportional to the general interest of each software application; if a problem is specific to a small and specialised user group, solving it may require more financial incentives to attract a wider developer community. As discussed in Chapter 1, free software developers are not motivated by a general interest in acting for the benefit of humanity, let alone the benefit of PAs. They are not volunteers providing charity. They are, by and large, strongly motivated through self-interest, and it is important for PAs launching a free software project to develop a clear idea of why participation in that project would interest developers. An understanding of developer motivations as summarised in this report will help. As a general rule, the more PA-specific an application is, the less attraction it will have for developers. A PA-specific application will also appeal to a smaller pool of developers – most likely, those professionally involved in PA-related work, including PA employees.

When there is less inherent attraction for developers in a given project, financial incentives are important. A project like GNU/LinEx in Extremadura can get away with distributing t-shirts to developers because of its high societal profile (and generic, rather than PA-specific technical aspect). Even in that case, the "seed" development was through pure financial incentives, in the form of a contract to a local software firm. In less appealing, lower-profile projects, direct financial incentives may be required. Innovative methods such as the "bounty" system discussed previously can be used, or direct methods such hiring developers as employees or issuing sub-contracts.

More generally, it is helpful to encourage SMEs and individuals to develop viable businesses for providing integration and support for the PA-specific applications they develop. This leads to an ecology that provides motives of (financial) self-interest for developers to voluntarily contribute to the development of PA-related software in the expectation of revenues from services provided to maintain or support such software. These incentives were partially responsible for the creation of Governmentforge.org and Schoolforge.org, by organisations providing services and support to governments and schools. Such encouragement to SMEs or contractors need not be at a large scale. For instance, when a unit of a particular PA designs a small, PA-specific software application with the support of a particular subcontractor, the PA could actively promote the use of the resulting software in its departments and in other PAs. This helps build a market for the contractors to provide support, reducing the need for direct financial incentives for further maintenance and development on the project.

j) Monitor and evaluate results

Clearly it is important to continuously monitor the extent of interaction between "client" PAs and the "supplier" communities (although the suppliers are likely to include PAs or at least individuals from within PAs). When interest dies, projects can die and fail to have a sustainable maintenance model. It is important therefore to promote continuous interaction between users and developers in order to ensure sustainable projects. This interaction can be evaluated through a number of indicators, including the degree of community participation (number of developers, frequency of contribution, amount of on-line discussion, number of downloads etc); and the degree of responsiveness to PA interests (time lag between bug reports and bug fixes; number of technical suggestions from the PA that are actually adopted and implemented by the community). When these indicator show flagging community support, PAs may need to increase feedback (f), improve their identification and recognition of leaders (g) and quite possibly increase financial support (i).

k) Involve other public administrations

Finally, as was recommended in the IDA POSS study[10], PAs should work in groups, as single PAs are much less likely to succeed in generating a community of supporting developers. This is because, as described previously, a community of supporting developers is attracted by an existing community. Getting like-minded PAs together before approaching a wider community therefore is more likely to succeed than a lone PA trying to form a community around itself. There are of course exceptions that prove this rule, such as Extremadura, but that required extensive high-profile support at the highest level of government to succeed – with the expressed intention of changing the society in an entire region. PAs with less ambitious goals, such as developing an efficient system for managing personnel records, may need to find other like-minded PAs to form coalitions before taking initiatives to engage with the developer community.

[10] www.europa.eu.int/ida/oso - Resources

The following chapter outlines the legal issues involved in the collaborations PAs must build, between PAs and developers and between all the authors involved, i.e. the licensing environment.

5) Legal relationships

a) Identify the relationships

To allow public administrations partnering with free software developers, there are basically two kind of relationships: 1) between the administration and the developers, including the general relations between the authors of the project, and 2) between the authors (copyright owners or licensors) and the users (licensees) including the fact that users do not necessarily belong to PA and may want to use, reuse and sometimes redistribute the software according to their own vision or business objectives.

The previous sections have identified the relations that were “possible” (in a sense of “able to produce an expected result”) and the relation that were not (in a sense of “probably based on un-realistic forecastings and incapable of producing any concrete result).

We have now to summarise how the above relations may be translated in contracts and in choosing the appropriate license.

The first question is to identify the “type of software solution” (generic or specific to the needs of the PA), the “type of service” desired by the administration (new development from scratch, improvement of an existing software, integration and operational support of existing components), and the “type of partner” (the possible open source communities or other partners able to deliver the service needed to obtain the solution).

1) Type of software solution

A1	Very generic purpose (e.g. operating system, desktop environment, Office suite)
A2	Specialised purpose (e.g. a content management, a collaborative work environment, a workflow system that could be used by various PA but also for other business)
A3	Specific PA purpose (e.g. land record management, public health)

2) Type of service

<u>B1</u>	Development from scratch
B2	Improvement of a solution developed externally (e.g. with the purpose to reuse it, to adapt it to local needs, to reach a specific security certification or to integrate components into a specific distribution)
<u>B3</u>	Improvement, take over or support of a solution developed internally by the administration

3) Type of partner

<u>C1</u>	“open source” or “free software” community without formal organisation or legal personality
C2	Organised community (e.g. non profit organisation of users or developers with legal personality)
<u>C3</u>	Commercial partner (a company or a consortium, specialised in development or integration of software solutions)

b) Possible cases, and corresponding contractual relations

In practice, all type of solutions, services and partners will not produce (3x3x3) 27 different relationships. To simplify, we believe that mainly the five case illustrated below (case 1 to 5) may be encountered.

A1	B1	C1/2/3	Not usual, outside exceptional circumstances as a voluntarism long term development	N/A
A2 or A3	B1	C3	To develop from scratch it is recommended to define detailed specifications, and to contract with a solid responsible third party (with development capabilities). Choice of standards, “open source conditions” and license to include/add into the development contract	Case 1
A2	B2	C1 or C2	In this case the solution has already been developed externally and has a “community” of developers and of users. There is already a licensing policy. This case has theoretically the most chances of success.	Case 2
A3	B2	C1 or C2	This is the “Pooling” hypothesis: a specific software has been developed by another administration and your PA wants to reuse it (or at the opposite other PA’s request to reuse/adapt your solution)	Case 3
A3	B3	C1 or C2	The problem here is to organise a community that is willing to endorse the specific needs of the PA. This will be possible provided the PA	Case 4

			stays strongly involved (providing both community members and incentives or bounties)	
A3	B3	C3	If the PA wants to “preserve the heritage” and ensure (e.g. for political reasons) that investments done by itself in the past will stay as an “asset” for the FLOSS community, but at the same time has no human resource to stay involved in development.	Case 5

i) Case 1 (solution developed from scratch for the PA)

The PA should commission a service company to develop the solution (or part of it). The principle decision to distribute the solution as open source software must be taken from scratch (if this distribution is “possible” – even if it is finally not done), and the selection of the distribution license too, because this conditions the possible components and standards (non-proprietary, with a compatible license) that the service company may use to realise the solution. If the code is entirely written from scratch, the PA is entirely free about the choice of a licence. At the contrary, if the developers are allowed to reuse and integrate existing pieces of free software (no reinventing the wheel) the license of these components will determine (in the case of *copyleft*[11] effect) the condition of re-distribution of the developed software.

The RFP (request for proposal) will therefore:

[11] In the aim of avoiding later appropriation „copyleft“ clauses are license dispositions that generally mandate the use of the initial software or component license if and when redistributing any new software integrating all or part of these initial software or component.

- Indicate the license chosen by the administration in case the administration wants to distribute the solution (see hereafter)
- Check and clarify all copyrights questions:
    - The moral rights (to the “brand name” of the solution, to disclosure of rights and to the respect and integrity of the work that has been done)
    - The propriety rights:
        - who can “represent the solution” on web sites, events and other communications
        - who can reproduce (as is), modify, redistribute,
        - what about redistribution fees (as in general no open source license forbid to sell the redistribution…).

The recommended best practice (clause to include in the RFP) is that:

1) The PA has the exclusive right of disclosure and the proprietary rights of representation and reproduction of the software.
2) The service provider keeps the right to be identified as the author. However, he can only reuse the results, commercially or free of charge, with the entity’s express permission, so he cannot automatically redistribute freely: the PA keeps the decision to redistribute or not.

- Determine the development and interoperability standards (highly recommended). If a precise indication of all standards cannot be done, precise that any standards must be:
    - Non proprietary or at least licensed for free, for any use or redistribution

    - Endorsed by a neutral, acknowledged international (or European) normalisation association or specialised administration.
- Impose to the service partner to use only software components and standards that are fully compatible with the above (e.g. your service partner cannot integer GNU/GPL components if the PA has decided to distribute under a BSD license). The service partner must issue a certificate to confirm this conformity.

ii) Case 2 (Specific open source software, found externally)

In this case the PA is looking for using and adapting an existing FLOSS project, that has already a community of developers (and if delivered, of users), responding to a specific purpose (e.g. a content management, a collaborative work environment, a workflow system that could be used by various PA but also for other business)

- Identify the community and its leaders
- Identify the licenses and check that it is conform to the PA needs (negotiate appropriate license if possible).
- Deploy contractual or extra-contractual incentives:
    - Contracting a service level agreement ensuring support (this is only possible if the author is an organised body or a company)
    - “Ex Ante” funds to orient developments in the desired direction (this is also only possible if the counterpart is an organised body or a company)
    - Deploying incentives (especially if the supporting community is not an organised legal body):

- Provide Job opportunities (temporary or definitive engagement of one or more community experts)
- Create an “Ex post” bounty or award system in case of improvement and/or support to the PA objectives (e.g. let the Open source community create a committee or scientific council that will assess the value of contributions in regard to the PA objectives and will recommends the “ex post” awards)
- In this case the solution has already been developed externally and has a “community” of developers and of users.

iii) Case 3 (solution developed by a PA, with possible pooling or reuse)

As said within the POSS study published on the IDA site, the decision of distributing solutions under open source license by the PA creating it must be made on a case-by-case basis according to the benefits of the software for a group of users. Going open source in a multi-lingual environment may represent an important extra investment (e.g. multiply by 3 the initial cost) and if the software does not clearly meet the needs of other PA’s, it is preferable not to bear the extra burden of distribution.

The process will be, first:

- Choosing the license in case the administration wants to distribute the solution (see hereafter). This has to be done from scratch as it has an impact on the components that developers may reuse or not.

- Assemble full information and certify license compliance for all use of FLOSS components during the project. If these components are used, document carefully their license, as the actions to comply with it.
- At the end of the project, consolidate the compliance report and indicate the name of the server on which the software is distributed as free software, the date when it will be made available.
- Notify the software existence and copyright to a competent copyright organisation and to the FLOSS organisation as the Open source or the Free Software Foundation, (especially when the PA wants to benefit from support coordinated by that organisation).

iv) Case 4 (Specific solution, developed internally by the PA, so far)

The PA has already developed a solution with a specific public sector purpose (e.g. land record management, public health) and wants the solution to be take over by an Open source community, ensuring pereniality, improvements and support.

Prior to the decision of going open source, the PA should take contact with national / European free developers organisations and explore if there is some guarantee that an existing community will be interested and endorse the project. If this is not clear, the recommended solution is to act as in case 5.

After choosing a license (in consideration with the components integrated into the solution - see hereafter, and the philosophy of the above community), the PA should actively participate in the reinforcement of the community, possibly in collaboration with other PA sharing the same needs.

A rapid “de-investment” is not possible here.

Once the community is created or identified, a similar (stronger) incentive policy as explained in case 2, should be implemented.

v) Case 5 (same as 4, with a service partner)

The alternative to case 4 is, in addition to appropriate open source licensing, to contract long term (2 years and more) a service level agreement with a service provider at the same condition as for case 1. A part of the service provider mission will be to animate a contributor's community and set up the appropriate tools for it (therefore, chose a partner having experience in such co-operation). The cost will be higher than for case 4, but obtaining a result will be more simple and secure – at least concerning support, although without guarantee to generate long term a successful open source product supported by a community of volunteers.

c) Choosing the licence

If it is decided to distribute free/open source software, the licence must be chosen from the start of the project, when the specifications are drawn up, as the choice of components that can be reused during production will depend on the choice of licence.

As known there are (on the Open source Initiative web site) more than 30 licenses that are compatible OSI conditions. As said in the POSS study, these nine conditions were established by the Open source initiative (OSI – Bruce Perens[12]) to accept a license as "Open source". These nine conditions are the base of the OSD (Open source Definition):

1. Free Redistribution

[12] Bruce Perens wrote the first draft of this document as "The Debian Free Software Guidelines", and it is now a cornerstone of the OSI policy- see at http://www.opensource.org/docs/definition.html

The license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programmes from several different sources. The license shall not require a royalty or other fee for such sale.
Rationale: *By constraining the license to require free redistribution, it eliminates the temptation to throw away many long-term gains in order to make a few short-term sales dollars. Without that, there would be lots of pressure for co-operators to defect.*

2. **Source Code**
The programme must include source code, and must allow distribution in source code as well as compiled form. Where some form of a product is not distributed with source code, there must be a well-publicized means of obtaining the source code for no more than a reasonable reproduction cost–preferably, downloading via the Internet without charge. The source code must be the preferred form in which a programmer would modify the programme. Deliberately obfuscated source code is not allowed. Intermediate forms such as the output of a pre-processor or translator are not allowed.
Rationale: OSI requires access to un-obfuscated source code because you can't evolve programmes without modifying them. Since OSI purpose is to make evolution easy, it requires that modification be made easy.

3. **Derived Works**
The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.
Rationale: The mere ability to read source isn't enough to support independent peer review and rapid evolutionary selection. For rapid evolution to happen, people need to be able to experiment with and redistribute modifications.

4. **Integrity of The Author's Source Code**
The license may restrict source-code from being distributed in modified form *only* if the license allows the distribution of "patch files" with the source code for the purpose of modifying the programme at build time. The license must explicitly permit distribution of software built from modified source code. The license may require derived works to carry a different name or version number from the original software.
Rationale: Encouraging lots of improvement is a good thing, but users have a right to know who is responsible for the software they are using. Authors and maintainers have reciprocal right to know what they're being asked to support and protect their reputations.
Accordingly, an open-source license must guarantee that source be readily available, but may require that it be distributed as pristine base sources plus patches. In this way, "unofficial" changes can be made available but readily distinguished from the base source.

5. **No Discrimination Against Persons or Groups**
The license must not discriminate against any person or group of persons.
Rationale: In order to get the maximum benefit from the process, the maximum diversity of persons and groups should be equally eligible to contribute to open sources. Therefore OSI forbids any open-source license from locking anybody out of the process.
Some countries, including the United States, have export restrictions for certain types of software. An OSI-conformant license may warn licensees of applicable restrictions and remind them that they are obliged to obey the law; however, it may not incorporate such restrictions itself.

6. **No Discrimination Against Fields of Endeavour**
The license must not restrict anyone from making use of the programme in a specific field of endeavour. For example, it may not restrict the programme from being used in a business, or from being used for genetic research.
Rationale: The major intention of this clause is to prohibit license traps that prevent open source from being used commercially. OSI want commercial users to join OSS community, not feel excluded from it.

7. **Distribution of License**
The rights attached to the programme must apply to all to whom the programme is redistributed without the need for execution of an additional license by those parties.
Rationale: This clause is intended to forbid closing up software by indirect means such as requiring a non-disclosure agreement.

8. **License Must Not Be Specific to a Product**
The rights attached to the programme must not depend on the programme's being part of a particular software distribution. If the programme is extracted from that distribution and used or distributed within the terms of the programme's license, all parties to whom the programme is redistributed should have the same rights as those that are granted in conjunction with the original software distribution.
Rationale: This clause forecloses another class of possible problem, avoiding that the license forbids or restricts rights to use other programmes or at the contrary impose to use other programmes (e.g. included in the same distribution).

9. **License Must Not Contaminate Other Software**
The license must not place restrictions on other software that is distributed along with the licensed software. For example, the license must not insist that all other programmes distributed on the same medium must be open-source software.
Rationale: Distributors of open-source software have the right to make their own choices about their own software.
According OSI, the GPL license is conformant with this requirement, as GPLed libraries "contaminate" only software to which they will actively be closely linked at runtime, not software with which they are merely distributed.

d) Recommendations concerning the license

A first option is to try to make an original licence that answers specifically to the PA needs. In general, we do not recommend this option for several reasons: First, there is a risk that the PA's legal service, unaware of the specificity of open source matters, will over-protect or introduce conditions that will make the license simply not a "free or open source" one: for example, a license that reserves the use of the distributed software to other public administrations or exclude commercial use is not a FLOSS license.

Second, open source communities are usually "attached" to their licensing model. There is a risk that nobody will support your project if you make your own "exotic" license for the above reason and because reusing existing components (e.g. licensed under GPL license) may not be compatible anymore.

Therefore, a new license should be introduced only if it presents substantial advantages above all existing ones (e.g. a variant with better conformity to European laws) and if there is a strong and clear policy to generalise the use of this license to many software (and not to use it only in the specific case).

Where no such ambition exists to create and promote long term a new general license, the best option is therefore to join a "dominant license" supported by a large proportion of the open source community.

Two main options may be reasonable:

- If the PA wants to "give" its software for all type of re-use, including the incorporation in proprietary commercial solution (with the purpose of facilitating software business or software industry), a **BSD** type license should be chosen. This will not impeach the open source community to distribute the original code freely, and to fork its own improved versions under a GPL license (for example). However, improvements and new versions created by the proprietary industry may definitively escape to the open source world.
  In addition, this choice of a licence where "appropriation is tolerated" exclude the use of GPL components (about 50% of available components) and would mean retaining only components under licences in the same family – tolerating appropriation (BSD, MIT, Apache, Python, Zope, etc).

- If the PA wants to reserve its software for open source use (this does not restraint any form of commercialisation, but it restraint exclusive appropriation), then the **GPL** or a *copyleft* license fully compatible with the GPL is recommended (French open sources communities have recently promoted the CeCILL license, said to be more compatible with European law than the American GPL).
  The decision to produce a software development under a GPL license allows integration of components of GPL origin or nearly any other origin. It does not imply any obligation to distribute the software produced if it is produced entirely by the entity (to keep developments secrets, simply do not license this version).
  The use of the GPL may be less appreciated or questioned by some representatives of the industry, however it may ensure the largest volunteers support inside the open source community.

Appendix 1: Organisation, motivation and external interaction

e) How free software developer communities work

Based on an online survey of 2784 Free software/Free Software developers, the EC/FP5-supported FLOSS report provided insights into fundamental features of the OS/FS community and its economic principles. It shed light on the personal features of OS/FS developers, of their work and project organization, their motivations, expectations, and orientations.

The FLOSS community is a rather young and a predominantly male community with a strong professional background in the IT sector and a high educational level. Developers feature a high degree of mobility, and the European Union appears attractive mainly for developers from within its member states, but does not see a net inflow from developers from the United States of America or other world regions.

Overall, developing free software, with its strong voluntary incentive structure, resembles the structure of a hobby more closely than of salaried employment. Other than (software) engineers and programmers, students also play a significant role in the community, although project leadership and greater productivity tends to be the domain of experienced developers who earn income from their free software related work. Most of the developers participate in collaborative networks of a few people – informal teams of 3-5 are common. Nevertheless, a considerable large group of highly productive developers have regular contacts with more than 50 other developers and form what is undoubtedly the "professional elite" within the community.

These diverse independent developers come together to solve problems based on common interests; typically, they will work on modifying the software released by someone else, in order to adapt it to their own needs or fix problems with it; such adaptations are released into the community and absorbed into the next version of the software. Version changes occur rapidly and free software systems tend to grow and improve at a fairly rapid pace.

f) Organisation of free software developer communities

The development of software and the cooperation in free software developer communities differs considerably from developing proprietary software in firms. The free software model can be described as free cooperation of autonomous developers connected through a highly volatile network organization (David & Foray, 2002; von Hippel, 2002; Lakhani & von Hippel, 2003; von Krogh et al., 2003). Pinpointing the differences, proprietary software is usually considered to be aligned with a hierarchical organization of work, a sequencing of tasks, and a complex software architecture, whereas free software appears to be produced in horizontal networks in which various developers work at the same time at the same or similar tasks and with a modular software architecture. A significant advantage of the free software model to the proprietary software model its clearly its capacity in debugging, which results in unique software quality regarding stability and reliability. The vast pool of skilled people, flat hierarchies, synchronicity, and modularity of the free software model are considered as a significant means to counter the phenomena of the so-called software-crisis, i.e. that proprietary software often takes too long to develop, exceeds its budget, and does not working very well. (Feller & Fitzgerald, 2002) However, besides these fundamental differences free software development requires similar structures as proprietary software development, i.e. the members must have the capacity to write code, coordinate work, and to maintain and administer the advances of a project.[13] When a free software project reaches a critical size, individual people take response for different project tasks and coordinating institutions (steering committees) emerge. (Bauer & Pizka, 2003) The variety of such coordination mechanisms ranges from “benevolent dictators” (e.g. Linus Torvalds as leader of the Linux kernel development) over “rotating dictatorship” (e.g. the Perl scripting language) to democratically elected committees (e.g. FreeBSD, Apache, Debian (Raymond, 1998). How these institutions develop differs thus considerably between diverse projects and is yet not examined systematically, but technical merits and reputation (Garzarelli & Galoppini, 2003) and face-to-face contacts (O’Mahony &

[13] Additional capacities that must be provided are for instance an infrastructure to collect contributions and represent the status of a project, translating capacities, and legal expertise.

Ferraro, 2003) seem to play a significant role. In any case, public administrations that want to partner with free software developers can rely on the fact that the more a free software project is institutionalized, the more likely it is that they find persistent support and maintenance capacities in the free software community.

Regarding the structure of free software community members, a strong background in IT-related professions is its main characteristic. Software engineers and programmers provide roughly half of the community members; consultants and executives provide another 14 per cent (Students: 21 per cent, university staff: 9 per cent, others: 7 per cent). (Ghosh et al., 2004) Apparently, consultants and executives play a prominent role for the project activities in the free software community, as this group shows a strong activity with regard to project participation and leadership and regular contacts to other developers. Compared to this group, software engineers and programmers show lower activity degrees (Glott, 2004). In conclusion, free software developer communities do not only provide technical but also managerial expertise to organizations that have an interest in partnering with these communities.

g) Motives of free software developers

There is a large variety of assumptions as to why people join the FLOSS community and "work for free". Lakhani and Hippel (2000) distinguish three kinds of motivators: the personal need for using a specific free software application (see also Searls, 1998), the wish for reputation among developers (see also Raymond 1998a, 1998b; Kohn, 1987), and intrinsic joy in programming.[14]

Other authors point out that direct or indirect rewards are as essential for the functioning free software as for any other organisation that develops and distributes software. Ghosh (1998a) explains that developers always get more out than they possibly can put in, but only if people keep contributing all together. Implicitly he thereby

[14] Shah (2003) emphasises a specific need for software as the main reason for peoples' initial involvement, while on the long run fun becomes a more important motivation for people to continue.

assumes reciprocity on a large scale (see also Ousterhout 1999). Lerner & Tirole (2002; see also Lee, Moisa & Weiss, 2003) argue that the whole free software phenomenon can be explained by "simple economics", whereby they refer mainly to monetary and career concerns. Indeed, as the FLOSS survey turned out, more than half of the sample (54 per cent) earns either directly or indirectly money from free software. Both kinds of rewards seem to have the same importance for the community. Within the scope of directly earned money from free software, administrating software plays a more important role than developing software. Within indirect earnings, to get a job because of expertise in free software is observably the most important factor.

However, empirical research turned out that motivations for participation in free software projects are more complex. Hars and Ou (2001) observe a mixture of internal motivators, such as self-determination, altruism, and community identification, and external motivators, such as selling products, human capital improvements, self-marketing, and peer recognition. Further empirical analyses of free software developers' motives revealed that signalling functions play only a very unimportant role compared to other motivators.[15] Bonnaccorsi & Rossi (2003), who have compared several of such empirical studies, could show that in all these studies social motivations played a much more important role for free software developers than signalling effects or peer recognition (see also Ghosh 2003: 12-15).

In the FLOSS developer survey (Ghosh et al., 2002), almost eight out of ten software developers started with free software because they wanted to learn and develop new skills, and half of the sample claimed that they wanted to share their knowledge and skills with other software developers. The second important group of motives ranges from socialising (participation and collaboration) over software-related (wish to improve software products) to political (attitude against proprietary software) aspects of the community. It is noteworthy that all these reasons gained importance after the developer

[15] These studies are: Ghosh, Glott, Krieger & Robles (2002); Lakhani, K. R.; Wolf, B.; Bates, J. & DiBona, C. (2002); Hars and Ou (2002); Hertel, Niedner & Herrmann (2003).

had joined the community and attained some experience. Material motives ("improve my job opportunities", monetary concerns) were less important, although the share of those who wished to make money increased with growing experience in the community (from 4% as a motive to join the community to 12% as a motive to stay in the community). Conclusively, the initial motivation for participation in free software aims at individual skills and the exchange of information and knowledge with other developers, but over time material and political aspects grow.

Based on a factor and a cluster analysis of these data, Ghosh et al. (2004) could identify six diverging groups of developers from the initial motivations (that are reasons to join free software) and four diverging groups from the continuing motivations (that are reasons to continue free software)[16]. Apparently, most people who join the community have no clear concept of what they expect from this step and what its outcome will be, as almost half of the respondents of the FLOSS developer survey showed very diffuse motivations. "Ideologists" provided the second largest initial motivational group (17 per cent), which highlights the importance of the political dimension of the FLOSS community for attracting new members. Material motivations, reputation, and software-centred motivations (the wish to improve software products) showed similar shares between 10 and 12 per cent, whereas "enthusiasts", who showed a strong interest in almost all aspects of free software, provided clearly the smallest initial motivational group (4%).

Regarding the continuing motivational groups, Ghosh et al. (2004) observed a dissolution of the initial "enthusiasts", "materialists", and "diffused" to the benefit of a new group that is strongly driven by the wish for skills improvement, while "ideologists", "recognition seekers" and "software improvers" remained. "Skills improvers" (33 per

[16] N.B.: The term "initial motivational team" does not mean that members of this group belong to starters in the community. The term only describes a group within the FLOSS community that reported a certain set of motivations to join the FLOSS community that could be distinguished from other groups with different initial motivations. The same applies to the term "continuing motivational groups", which is not meant as a label for people who belong to the FLOSS community for a longer period. The initial motivation gives thus answer to the question "why did you join the FLOSS community?" (disregarded whether this was five weeks or five years ago), whereas the continuing motivation answers the questions "why are you still in the FLOSS community?" (disregarded whether one belongs to the community for five weeks or for five years).

cent) and ideologists (31 per cent) provided the two largest continuing motivational groups, whereas “software improvers” provided a quarter of the sample. “Recognition seekers” clearly provided a minority (12 per cent).

What do these diverse motivations present within the pool of potential contributors to a free software project mean, with regard to guidelines for public administrations’ involvement? For an IT department in a public administration, it would be best to somehow attract developers from whom software improvement and recognition play the most important role. In fact, skills improvement and ideology are the most important driving forces of free software developers, but this is not a disadvantage. In contrast, these aspects provide the foundation on which expertise in free software can be generated. In conclusion, public administrations that want to partner with free software should pay attention to the developers’ interest in challenging tasks and to their values and beliefs with regard to sharing knowledge, information, and software code / products. Given that, it is most likely that developers will be attracted towards software projects of immediate benefit to them – which is not necessarily the case for many dedicated public administration applications (e.g. for land records) but may well be the case for more general platforms used by public administrations (e.g. Zope or KDE). Developers may also be attracted to projects that provide the opportunity to enhance their reputation among the peers or the general public – including potential employers. Schemes to provide highly visible credit to contributors to public administration projects may therefore attract developers, providing that they are also technically challenging.

h) How free software projects interact with external feedback

Interacting with funding is a subset of a broader model whereby free software projects tend to be open communities, continuously responding to external feedback. Indeed, these communities do not relate to external feedback as much as absorb external commentators as part of a continuously growing community. Linus Torvalds, the creator of the Linux kernel that forms part of the GNU/Linux operating system, said as far back as in 1996 that “the large user-base has actually been a larger bonus than the developer base, although both are obviously needed to create the system that Linux is today. I simply had no idea what features people would want to have, and if I had continued to do

Linux on my own it would have been a much less interesting and complete system" (Ghosh 1998b).

There is widespread recognition in the free software community that users play a crucial feedback role. This is tempered, though, with the assumption that those who want a solution badly enough are free to (and therefore should) create it, or modify existing software to achieve it themselves. The degree of responsiveness to user suggestions is therefore likely to be directly proportional to the extent of the perceived user need – if a solution is likely to significantly improve the experience of all users (e.g. a bug that makes an operating system crash) it is likely to get fixed much more quickly by the community than something that satisfies a relatively small user base (such as a water management system). This is for the simple reason that communities work as collections of individuals, and there is a high chance of one (or more) individual participants taking the initiative to solve a given problem if it seems to be an important problem. Issues that affect specialised users only are likely to be solved based on more initiative by those users themselves.

For public administrations wanting to collaborate with free software communities, therefore, this means that active participation in getting major projects to work more in line with needs of PAs can work, when PAs are a big user base and the solutions are not extremely specialised. This is, for instance, quite common in free software projects like MMBase (especially in the Netherlands where a number of PAs use it and a number of Dutch businesses have been built around it) or Zope. For more specialised needs, however, the strategy must be more along the lines of attracting a community of interested parties around the PA's specific needs.

i) Paying for free software

Firms' interest in free software is not only reflected in their demand for free software products, but also in their direct support through either hiring people with free software experience or by providing personal expertise, as indicated by the role of consultants and executives in the free software developer community. As shown in the above section, 54% of the community members receive direct or indirect monetary rewards from free software. However, since the FLOSS developer survey did not ask the respondents to quantify how much they earn from free software and since other studies in the meantime did not ask this question either, it is not possible to provide a well-reasoned estimate of the monetary value of free software in terms of salaries or other remuneration that are paid for free software by external organisations.

Still, the costs for some free software products and related services can be quantified. Although it is often said that free software is distributed for free or at marginal costs, there are some free software products and services that appear quite expensive. For instance, a professional licence from MySQL costs 500 Euro, and entry-level technical support costs another 2000 Euro.[17] However, these costs are low compared to costs for proprietary software licences and service packages and it must be noted that the “dual licensing” model of MySQL AB requires these fees only for commercial use of the MySQL database that would violate the terms of the GPL free software licence. Most non-commercial use, and indeed most commercial use is permitted according to the GPL, so the same software can be downloaded at no cost, subject to the free software licensing terms, and without guaranteed technical support. Thus, besides the clear interest in profit, fees in free software may also serve as a system for keeping the free software model alive and putting off commercial third-party appropriation of the software.

Nevertheless, savings in software licence fees and support are only a small part of the factors that may serve as incentives for institutional users of free software, since participating in free software appears sensible for many reasons. For instance, firms can make money from offering complementary services to software (Wichmann, 2002a; Feller & Fitzgerald, 2002), like the Linux distributor Red Hat[18]. But there are many other incentives that apply also to public administrations and that result in funding free software. They can

- cut hardware costs because free software systems do not in general require as frequent hardware upgrades as proprietary software (Feller & Fitzgerald, 2002)

[17] See https://order.mysql.com/

[18] http://www.redhat.com/

- become independent from powerful proprietary software companies and related high prices and licence fees for software (Lerner & Tirole, 2002a),
- cut innovation, training, and maintenance costs by tapping the (almost costless) development and consultancy capacities of the free software community (von Krogh et al., 2002; Hawkins, 2002)
- can hire experienced and skilled developers from the free software community, i.e. reduce search costs for good personnel (Wichmann, 2002b; Fink, 2003)

Accordingly, Ghosh & Glott (2003) have found similar incentives for the Dutch public sector, of which the availability of the source code, the capacity of free software to be combined with proprietary software, the saving of software- and IT-related costs, and the fact that free software can easier be customised than proprietary software have been most important.[19]

[19] However, there were also strong disincentives: Public administrations often doubt whether they would find technical support for open source software and they fear high training costs.

Appendix 2: A closer look at selected development projects

j) FreeBSD

There exist other free operating systems besides the popular GNU/Linux. A family of them are the successors of the distributions made at the University of California at Berkeley: the BSD systems. The oldest and most-known of these systems is FreeBSD, whose history goes back to the beginnings of 1993 when Bill Jolitz stopped releasing the non-official patches to 386BSD, the system officially released by Berkeley. With the assistance of a company called Walnut Creek CDROM, which was later on renamed to BSDi, a group of volunteers decided to continue with the effort of keeping this operating system up-to-date.

The main goal of the FreeBSD project is to create an operating system that can be used without any obligations or hindrance, but with all the advantages of the availability of source code and of a high-quality development process. The FreeBSD is released under the terms of a permissive BSD license which allows the redistribution of itself or of modified versions even in a proprietary way. This licence is almost the same as public domain (release free from copyright).

i) History of FreeBSD

The 1.0 version appeared ending 1993 and was based in 4.3BSD Net/2 and 386BSD. 4.3BSD Net/2 contained code from the seventies when UNIX was developed by AT&T, resulting in legal problems that were not solved until 1995, when FreeBSD 2.0 was released with no code being originally from AT&T. Rather, this was based on 4.4BSD_lite, a “light” version of University of California's 4.4BSD that had suppressed many modules to avoid legal problems and that had an incomplete port to Intel systems.

The history of FreeBSD would not be complete if we did not talk about its “brother” distributions, NetBSD and OpenBSD. NetBSD appeared with a 0.8 version in mid-1993. Its main goal has been to be very portable although in the beginnings it was only a port to the Intel 80386 platform. Its slogan is “of course it runs NetBSD”.

OpenBSD is the product of a fork from NetBSD based on philosophical (and personal) differences between developers in mid-1996. Its main focus is on security and cryptography - it is said to be the world's most secure operating system (with the possible exception of SE Linux developed by the US National Security Agency), although as it is based on NetBSD it conserves a great portability.

ii) Development in FreeBSD

The development model used in the FreeBSD project is based strongly in the use of two tools: the CVS versioning system and the GNAT bug-tracking system. Hierarchies in the project are based on them, too. This way, committers – those developers who have write access to the CVS repository – have control over the project acting as "gatekeepers" at least to the "official" release. It is always possible, this being free software, that rebellious programmers can create and control their own version, which is called "forking" a project and is what resulted in NetBSD and OpenBSD.

There is no need to be a committer to submit bugs to GNATS, so anybody may submit one. All open contributions in GNATS are evaluated by a committer who may assign the task to other committers or request more information to the person who did the original report. If the bug has already been solved in one of the most recent branches of development, its status is updated. In any case the goal is that the report of each bug is tracked, and the tracking is "closed" at the end of the process once a bug is fixed.

FreeBSD delivers its software in two different forms. On the one hand we find the ports, a system of downloading the source code, compiling and configured to work on different machines. On the other hand there are packages, which are the same source codes as the ports but precompiled and hence available for download in binary form. The most important advantage of ports over packages is that they allow the user to configure and optimize the software for his own machine while the system based on packages allows one to install the software faster as it comes in a precompiled form.

iii) Decision-taking in FreeBSD

The board of directors of FreeBSD, popularly known as the “core team” is the one that points the direction the project should follow and that looks that its goals are achieved. It is also supposed to mediate in case of conflicts between committers. Until October 2000 it was a closed group which one only could join by invitation. Since then, the members of the core team are periodically and democratically elected by the developers.

iv) Companies around FreeBSD

There are several companies that offer services and products based on FreeBSD. The FreeBSD project has a detailed list of them on its web site. In this short study we will focus on two: BSDi and Walnut Creek CDROM.

FreeBSD was born partly because some people at CSRG (Computer Systems Research Group) at the University of Berkeley founded in 1991 a company named BSDi. This company gave commercial support for their new operating system. In addition to the commercial version of the functional FreeBSD system, BSDi also developed other products such as an Internet server and a gateway.

Walnut Creek CDROM was born with the objective of commercializing FreeBSD as a finished product, a sort of distribution such as the GNU/Linux ones, but based instead on FreeBSD. In November 1998, Walnut Creek opened the FreeBSD Mall web site that offered all type of products related to FreeBSD (from the distribution itself to T-shirts, magazines, books, etc.) and offered professional support for FreeBSD.

March 2000, BSDi and Walnut Creek merged under the name BSDi to confront the Linux phenomenon which was leaving the BSD systems in the shade. A year later, Wind River bought the part dedicated to software development at BSDi with the intention of adapting FreeBSD to embedded systems and intelligent devices connected to the Net.

v) Statistics of FreeBSD

The numbers that will be shown next correspond to the analysis of FreeBSD's CVS records on the 21 of August of 2003. SLOC refers to source lines of code, and COCOMO is a standard model in software engineering used to estimate the cost of development of a software product. Given the product's expected size in SLOC, COCOMO enables firms to estimate the number of person-years (effort) and the actual number of years (duration) that the project would require to complete. This can be used to estimate costs of development based on average developer salaries. The figures based on COCOMO in the following tables show for each free software project the investment that could have been required if the software had been developed within a proprietary software company.

Table 1. FreeBSD

Web site	http://www.FreeBSD.org
Project start	1993
Licence	BSD-type
Current version of FreeBSD	4.8 (stable), 5.1 (development)
Lines of code (SLOC)	7,750,000
Number of files	250,000
Cost estimation (COCOMO)	$ 325,000,000
Development time estimation (COCOMO)	10.5 years (126 months)
Estimation of the mean number of developers (COCOMO team size)	235
Approx. actual number of developers	400 commiters (1000 developers)
Active commiters in the last year	75 (less that 20% of the total amount)
Number of commits in CVS	2,000,000
Mean number of commits per day	approx. 500
Main tools	CVS, GNATS, mailing lists, news sites (web)

k) KDE

Although certainly not the first solution regarding user-friendly desktop environments, the spread in 1995 of the Windows95 operating system led to a radical change in the interaction between computers for common users. The followers of UNIX perceived the success of Windows95 as something lacking in the UNIX world (which was originally the home of one of the earliest graphical user interfaces, the X Window System) and launched several efforts in order to have a desktop environment for the free software world. In 1996 KDE – K Desktop Environment, where K once stood for “Kool” – was born.

A big public debate arose when the newly created KDE project decided to use an object-oriented library called Qt, developed by Norway-based firm Trolltech(TM), and which was not released under any free software license. Hence, we had a situation where although the applications developed by the KDE group were free software (they chose the GPL license), they linked to this Qt library making the environment as a whole impossible to be redistributed, as one of the “four freedoms” of the free software definition (freedom to redistribute) was violated. From the KDE version 2.0 Trolltech released Qt under a dual license that specifies that if the application that links to it is GPL, Qt is licensed under the GPL. The licensing problem was solved successfully.

i) KDE development

KDE is one of the few free software projects that follows a strict shipping calendar. Versioning follows a defined policy: the first number gives the major version, the second is the minor version and the third (if it is given) gives the number of updated versions. Within versions with the same minor version number there exists binary compatibility, so it is possible to run an application in KDE 3.4 that was programmed for KDE 3.2. Generally changes in the major version have been done in parallel to new versions of the Qt library, allowing developers to make use of the enhancements in it.

KDE constituted itself later on in a registered association in Germany (called KDE e.V.) and as such it has some statutes in which a board of directors is specified. The influence of this board over the development is zero, as its tasks are related mainly to the administration of the association, especially regarding donations that the project receives. For the promotion and diffusion of KDE the KDE League was founded. It is formed by a group of enterprises and individuals. The companies that participate in the KDE League are mainly GNU/Linux distributors (SuSE, Mandrake, TurboLinux, Lindows and Hancom - a Korean-based free software distribution), development firms (Trolltech and Klarälvdalens Datakonsult AB), a giant company (IBM) and finally KDE.com which provides merchandising and other minor services related to KDE over the Internet.

ii) Statistics of KDE

Table 2. Statistics of KDE

Web Site	http://www.kde.org
Project start	1996
Licenses (for applications)	GPL, QPL, MIT, Artistic
Licenses (for libraries)	LGPL, BSD, X11
Lines of Code (SLOC)	6,100,000
Number of files	310,000 files
Cost Estimation (COCOMO)	$ 255,000,000
Development time estimation (COCOMO)	9.41 years (112.98 months)
Estimation of mean number of developers (COCOMO team size)	200,64
Approx. actual number of developers	around 900 commiters
Number of active commiters in the last two years	around 600 (65% of the total)
Number of commits in the CVS	approx. 2,000,000
Mean number of commits per day	1,700
Tools, documentation and events that help development	CVS, mailing lists, web sites,news sites, periodical meetings...

l) GNOME

The GNOME project has as primary aim to build up a complete, free and easy-to-use end-user desktop system. In addition, GNOME has pretensions of becoming a powerful development platform for the developer. GNOME is part of the GNU project and currently all its code base is released under a GNU GPL or GNU LGPL license.

GNOME is the response of some community members to the KDE licensing debates of 1997. Some developers, most notably Miguel de Icaza, started to develop an alternative desktop to KDE that would always be completely free (i.e. as in the freedoms of “free software”).

i) The GNOME Foundation

In October 2000, the GNOME Foundation was created. It is a non-profit organization but not a consortium of companies. It is responsible for coordinating releases, decides which subprojects form part of GNOME, and is the 'official' voice for the press and external organizations and promotes the project by means of conferences, standards and other activities. The GNOME Foundation also accepts funds to be used to sponsor its activities.

In general terms, the GNOME Foundation is structured with two boards: the board of directors and the advisory board. The board of directors is composed of at most 14 members elected democratically every year by all members of the foundation. Anyone who has contributed in some way (not necessarily software code, but also documentation, translation, etc) can become a member of the GNOME Foundation and hence have the right to vote. There are some restrictions placed on the board of directors in order to guarantee its transparency. For instance, at most four members of this board may be affiliated to a single company. In any case, an elected director performs as a private individual, not representing an organisation. This clause was introduced in order to ensure transparency of motives.

The other board of the GNOME Foundation is the advisory board which has no decision-making powers. It is formed by industrial partners as well as non-commercial organizations interested in the development of GNOME. Currently these are Red Hat, Novell, Hewlett-Packard, Mandrake, SUN Microsystems, Red Flag Linux (from China), Wipro (from India), Debian and the Free Software Foundation.

ii) Industry around GNOME

GNOME has achieved a substantial presence in the computer industry. Several enterprises are very active in its development and promotion. The most significant cases are Ximian Inc. (now part of Novell) and the no longer operational Eazel which were start-up companies founded with the aim of developing GNOME. RHAD Labs from Red Hat and recently SUN Microsystems have also about a dozen developers each devoted to this project.

iii) Statistics of GNOME

Table 1-3. Statistics of GNOME

Web Site	http://www.gnome.org
Project start	September 1997
License	GNU GPL and GNU LGPL
Lines of Code (SLOC)	9,200,000
Number of files	228,000
Cost Estimation (COCOMO)	$ 400,000,000
Development time estimation (COCOMO)	11,08 years (133,02 months)
Estimation of mean number of developers (COCOMO team size)	250 approx.
Number of subprojects	it has more than 700 modules in its CVS
Approx. actual number of developers	almost 1000 with write permission
Number of active commiters in the last two years	around 700 (75% of the total)
Number of commits to the CVS	1,900,000
Number of mean commits per day	around 900
Main development tools	CVS, mailing lists, web sites, news sites, yearly meetings...

m) Debian GNU/Linux

Debian is a free operating system that currently uses the Linux kernel to produce its distribution (although distributions with other kernels like The HURD are planned in the future). It is available for several computer architectures, including Intel x86, ARM, Motorola, 680x0, PowerPC, Alpha and SPARC.

Debian is not only the biggest GNU/Linux distribution currently; it is one of the most reliable and has been given awards several times by users and by technical publications. Although its user base is difficult to estimate as the Debian project does not sell CDs directly, and anybody has the right to redistribute it freely, it would be fair to say that it is one of the most important distributions in the GNU/Linux market. It is the basis for many “modified GNU/Linux” projects around the world (including GNU/LinEx in Extremadura, Spain) and according to the FLOSS Developer Survey, the most popular distribution in terms of use by developers themselves.

Besides its voluntary nature, the Debian project has a characteristic that makes it especially singular: the Debian Social contract. This document contains not only the primary objectives of the Debian project but also the means that will be used to achieve them. In Debian we can find a categorization of the packages by their license and the distribution requisites. The main part of the Debian distribution (the section called main which is composed of a big number of packages) contains only software that follows the Debian Free Software Guidelines (DFSG), which is part of the Social Contract.

Debian distributions are created by over a thousand volunteers (mostly IT professionals). The task of these volunteers is to take the source code of programs – of which in most cases they are not the original authors – and configure, compile and package them so that the typical user has only to select it a program in order to install it. This may be thought of as a simple task but is quite complicated as factors such as software dependencies have to be taken into account (for instance, when package A needs package B to work, a common occurrence) and the selection from various versions of all these packages.

The technical work that is undertaken by the members of the Debian project is the same that is realized in other distributions produced by companies (such as Red Hat or SuSE/Novell): software integration for its smooth functioning. In addition to the adapting and packaging tasks, Debian developers maintain an infrastructure of services that are based on the Internet (web page, mailing lists and their archives, bug tracking system, file download repositories, etc.), as well as other projects that are focused on translations and localization, the development of Debian-specific tools and in general any other element that makes the Debian distribution possible.

Debian is also known for having a strict package and versioning policy in order to achieve a better quality of the end product. Hence, at any given moment there exist three different flavors of Debian: a stable, an unstable and a test version. As its name indicates the stable version is the one that is targeted to systems and persons that do not want any surprises. Its software has to pass a quarantine period in which only bugs will be fixed and no developments or improvements made. The norm says that a stable Debian version cannot contain any known critical error. On the other hand, this makes the stable versions exclude the latest and more novel additions to the software.

i) *Statistics of Debian*

The last stable Debian version is Debian 3.0 which obtained the nickname Woody released in 2002. It contains 4,579 source code packages and around 10,000 binary packages. Debian is hence one of the biggest integrated software collections that exists.

Table 4. Statistics of Debian

Web Site	http://www.debian.org
Project start	16.8.1993
Licenses	The ones that comply with the DFSG
Current (stable) version	Debian 3.0 (alias Woody)
Lines of Code (SLOC)	105,000,000
Number of packages	4,579
Cost Estimation (COCOMO)	3,625,000,000 $
Development time estimation (COCOMO)	7 years
Estimation of mean number of developers (COCOMO team size)	around 4,000
Approx. number of developers (Debian maintainers/integrators only, not developers of individual packages)	Around 1,000
Main development tools	Mailing lists, bug-tracking system

n) GNU/LinEx

GNU/LinEx is the consequence of the political strategy of the Spanish region of Extremadura. It has been conceived with the aim of making Extremadura -one of the poorest regions in Spain and in the former EU15- a leading one in technologies related to the knowledge era. The efforts of the regional government are not limited to public administration and education as its intention is to make Linex been widely used also in the private sector. The project has been funded from two different sources: the first one was the regional government of Extremadura, while some funds were European Union FEDER (European Union Funds for Regional Development).

The history of GNU/Linex begins in 1998 when the regional government looked for a way of introducing the region into the new computing era without leaving anybody out of it. The cost calculations made not possible to use solutions that were introduced in that days. Instead the free software products offered an interesting -although then unexplored- starting point. The Linex project was created mainly because of economical reasons, but with time a shift of the project towards the fundamental ideas of the free software movement can be observed, branding the project finally GNU/Linex.

In the year 2000, the education administration was transfered by the Spanish central government to the regional government of Extremadura (Junta). A free software solution began to be studied in 2001 and the decision to make an own distribution was taken in November that year. The first version of GNU/Linex (version 2.0) was released March 2002 and was elaborated by a Spanish private startup company. Following versions were developed by staff from the Junta, associated organisations and volunteers. GNU/Linex 3.0 was released June 2002 and the last version GNU/Linex 2004 appeared in August 2004.

Technically GNU/Linex is based on the Debian distribution. This means that GNU/Linex is a modified Debian: the base system has been taken from Debian and software which has been identified as strategical for the aims of the project has been packaged and included, and default configurations have been made for the purposes of localisation (e.g. default usage of Spanish).

As GNU/LinEx is not an independent software project, it is not possible to make a statistical analysis similar to the above projects. In a sense, Extremadura has leveraged the tremendous existing and on-going investment in terms of time and effort of Debian contributors and benefits from having made a relatively affordable contribution: the Junta says it has spent only Euro 300 000 on the system, much of which is for material costs (printing and distributing manuals and CDs). On the other hand, the Junta claims to have saved over 1000 euro in software costs for every desktop set up with GNU/Linex and related software, rather than the proprietary equivalents. With an installed base of over 80 000, the savings are therefore significant. See the case study of Extremadura on the IDA Open Source Observatory (OSO)[20] for links and further information.

As GNU/LinEx is integrated with Debian as a localised version, it is not very meaningful to generate statistics as with the previously described projects.

[20] http//:www.europa.eu.int/ida/oso

References

Bezroukov, N. (1996): Portraits of free software pioneers. http://www.softpanorama.org/People/ (September 3, 2003).

Bonnaccorsi, A. & Rossi, C. (2002): Why Open Source software can succeed. Accessible online at http://opensource.mit.edu/papers/bnaccorsirossimotivationshort.pdf latest access September 3, 2003.

Bonnaccorsi, A. & Rossi, C. (2003): Altruistic Individuals, Selfish Firms? The Structure of Motivation in Open Source Software. http://opensource.mit.edu/papers/bnaccorsirossimotiva-tionshort.pdf (September 3, 2003)

Butler, B.; Sproull, L.; Kiesler, S.; Kraut, R. (2002): Community effort in online groups: who does the work an why? Accessible online at http://opensource.mit.edu/papers/butler.pdf, latest access March 22, 2003.

Castells, M. & Himanen, P. (2002): The Information Society and the Welfare State. The Finnish Model. Oxford: Oxford University Press.

Ciborra, C. U.; Andreu, R. (2001): Sharing knowledge across boundaries. In: Journal of Information Technology, No. 16, pp. 73-81

Dempsey, B. J.; Weiss, D.; Jones, P.; Greenberg, J. (2002): Who is an free software developer? Profiling a community of Linux developers. In: Communications of the ACM, Volume 45, No. 2 (February 2002); pp. 67-72.

DiBona, C.; Ockman, S.; Stone, M.: Introduction. In: DiBona, C., Ockman, S.; Stone, M. (eds.): Free softwares: voices from the free software revolution. Sebastopol, California: O'Reilly. Accessible online at http://www.oreilly.com/catalog/opensources/book/intro.html. Latest access: March 23, 2002

Ettrich, M. (2000): Wer kodiert? In: iX, No. 1/2000; p. 112. http://www.heise.de/ix/artikel/-2000/01/112/.

Feller, J. & Fitzgerald, B. (2000): A framework analysis of the open source software development paradigm. In: Proceedings of the 21st international conference on information systems, Brisbane, Queensland, Australia, p. 58-69.

Feller, J. & Fitzgerald, B. (2001): Understanding open source software development. Pearson Education.

Fitzgerald, B & Kenny, T. (2003): Open source software in the trenches: lessons from a large-scale OSS implementation. At Twenty-Fourth International Conference on Information Systems

Franke, N.; Shah, S. (2001): How communities support innovative activities: An exploration of assistance and sharing among innovative users of sporting equipment. In: Sloan Working Paper No. 4164, accessible online at http://opensource.mit.edu/papers/frankeshah.pdf, latest access August 7, 2002

Garzarelli, G. (2002): Open source software and the economics of organization. accessible online at http://opensource.mit.edu/papers/garzarelli.pdf, latest access August 7, 2002

Ghosh, R. A. (1998a): Cooking pot markets: an economic model for the trade in free goods and services on the Internet. Accessible online at http://firstmonday.org/issues/issue3_3/Ghosh/index.html; latest access March 26, 2002.

Ghosh, R. A. (1998b): Interview with Linus Torvalds on the economics of free software. Accessible online at http://firstmonday.org/issues/issue3_3/

Ghosh, R. A. and Prakash, V. V. (2000): The Orbiten free software survey. First Monday, vol. 5, no. 7 (July 2000), URL: http://firstmonday.org/issues/issue5_7/Ghosh/index.html; latest access March 26, 2002.

Ghosh, R. A.; Glott, R.; Krieger, B.; Robles, G. (2002): Free/Libre and Open Source Software: Survey and Study. Part IV: Survey of Developers. http://www.infonomics.nl/FLOSS/report/-Final4.htm. Maastricht: International Institute of Infonomics / Merit

Ghosh, R. A.; Robles, G. & Glott, R. (2002): Free/Libre and Open Source Software: Survey and Study. Part V: Source Code Survey. http://www.infonomics.nl/FLOSS/report/Final5all.htm. Maastricht: International Institute of Infonomics / Merit

Ghosh, R. A. (2003): Understanding Free Software Developers: Findings from the FLOSS Study. Paper presented at HBS - MIT Sloan Free/Open Source Software Conference: New Models of Software Development. June 19-20, 2003, Boston and Cambridge, MA

Hars, A.& Ou, S. (2001): Working for free? – Motivations participating in free software projects. In: IEEE (ed.): Proceedings of the 34th Hawaii International Conference on System Sciences.

Hertel, G.; Niedner, N.; Herrmann, S. (2003): Motivation of Software Developers in Free software Projects: An Internet-based Survey of Contributors to the Linux Kernel. Paper presented at HBS - MIT Sloan Free/Open Source Software Conference: New Models of Software Development. June 19-20, 2003, Boston and Cambridge, MA

Himanen, P. (2001): The Hacker Ethic and the Spirit of the Information Age. New York, Toronto: Random House.

Hippel, E. von & Krogh, G. von (2002): Exploring the open source software phenomenon: Issues for organization science. Accessible online at http://opensource.mit.edu/papers/hippel.pdf, latest access: September 3, 2003

Kohn, A. (1987): Studies Find Reward Often No Motivator. http://www.gnu.org/philosophy/motivation.html (September 3, 2003) [Reprint version, text was originally published in the Boston Globe, January 19, 1987]

Lakhani, K.& Hippel, E. (2000): How Open Source Software Works: “Free” User-to-User Assistance. MIT Sloan School of Management Working Paper No. 4117 (May 2000).

Lakhani, K. R.; Wolf, B.; Bates, J. & DiBona, C. (2002): The Boston Consulting Group Hacker Survey, release 0.73, in cooperation with OSDN. http://www.osdn.com/bcg/BCGHACKERSURVEY-0.73.pdf, latest access September 5, 2003

Lee, S., Moisa, N. & Weiss, M. (2003): Free software as a Signalling Device – an Economic Analysis. In: Department of Economics of the Johann-Wolfgang-Goethe University (ed.); Working Paper Series: Finance and Accounting, NO. 102, March 2003. Frankfurt/Main: Johann-Wolfgang-Goethe University.

Lerner, J.; Tirole, J. (2000): The simple economics of free software. NBER Working Paper Series, Working Paper 7600, National Bureau of Economic Research, Cambridge, Massachusetts, March 2000.

MITRE (2003): Use of Free and Open-Source Software (FOSS) in the U.S. Department of Defense. Prepared for the US Defense Information Services Agency. http://www.microcross.com/dodfoss.pdf; latest access October 29, 2004.

Moon, J. Y.; Sproull, L. (2000): Essence of distributed work: The case of the Linux kernel. In: First Monday, accessible online at http://firstmonday.org/issues/issue5_11/Moon/index.html; latest access March 27, 2002.

O'Reilly, T. (1999): Lessons from open source software development. In: Communications of the ACM, Volume 42, No. 4 (April 1999); pp. 33-37.Ousterhout, J. (1999): Free software needs profit. In: Communications of the ACM, Vol. 42/No. 4.

Raymond, E. S. (1998a): The cathedral and the bazaar. URL: http://www.tuxedo.org/~esr/wri-tings/cathedral-bazaar; latest access November 15, 2001.

Raymond, E. S. (1998b): Homesteading the noosphere. In: First Monday, vol. 3, no. 10 (October 1998), URL: http://firstmonday.org/issues/issue3_10/Raymond/ (September 3, 2003).

Scacchi, W. (2002): Understanding the requirements for developing open source software systems. Accessible online at http://opensource.mit.edu/papers/Scacchi.pdf, latest access September 3, 2003

Schmitz, P-E. (2002): Pooling Open Source Software. Accessible online at http://www.europa.eu.int/ida/oso (see Resources section)

Shah, S. (2003): Understanding the Nature of Participation & Coordination in Open and Gated Source Software Development Communities. Paper presented at HBS - MIT Sloan Free/Open source software Conference: New Models of Software Development. June 19-20, 2003, Boston and Cambridge, MA

Stallman, R. (1999): The GNU operating system and the free software movement. In: DiBona, C., Ockman, S., & Stone, M. (eds.): Free softwares: voices from the free software revolution. Sebastopol, California: O'Reilly. Accessible online at http://www.oreilly.com/catalog/opensources/book/stallman.html

Suriya, M. (2003): Gender issues in the career development of IT professionals: a global perspective. Accessible online at http://www.gisdevelopment.net/proceedings/mapasia/2003/-papers/i4d/i4d002.htm

Torvalds (1999): Interview with Linus Torvalds: What motivates free software developers? URL: http://firstmonday.org/issues/issue3_3/Torvalds/index..html; latest access March 26, 2002.

Wenger, E. (2000): Communities of practice – learning as a social system. Accessible online at: http://www.co-I-l.com/coil/knowledge-garden/cop/lss.shtml

Wheeler, D. A. (2002): Why open source software / free software (OSS/FS)? Look at the numbers! Accessible online at http://www.dwheeler.com/oss_fs_why.html. Latest access: June 14, 2002.

Young, R. (1999): Giving it away: How Red Hat stumbled across a new economic model and helped improve an industry. In: Free softwares: Voices from the free software revolution. Sebastopol, CA: O'Reilly & Associates.