Good practices for R&D projects producing FLOSS (Request for comments)

Jesus M. Gonzalez-Barahona
GSyC/LibreSoft (URJC), http://libresoft.es

v0.2, October 2010 (early draft)

Abstract

Many R&D projects are producing FLOSS (free, libre, open source software) in different domains. This document provides details on the practices that could be considered as "good" from the point of view of maximizing the impact of using FLOSS as a distribution model. It is targeted mainly at projects funded by the European Commission, but could be of interest or others as well.

Contents

1 Introduction

Deciding that an R&D project will produce some FLOSS (free, libre, open source software) product can lead to many beneficial outcomes. However, this decision alone does not necessarily produce them: it is important to take some actions to maximize the benefits, and to avoid the potential problems that could arise. This document provides some guidelines that could help to determine and understand those actions.

This document is not intended to explain the benefits or problems of deciding to go the FLOSS way. It is assumed that the decision of producing FLOSS has already been taken.

Although the main target of this document is R&D projects funded by the European Commission (EC) under its Framework Programme (FP), it could be of use probably for other R&D projects. In the case of FP EC funded projects, projects are usually carried on by a set of partners (the project consortium) who sign a Consortium Agreement (CA). The EC does not require (and usually does not want) transfer of copyright of the results of the project. On the contrary, they expect the project consortium to detail, in the CA, the intellectual and industrial property provisions they may agree. In this kind of projects partners can also publish or market the results of the project, or integrate them with other products, with the only constraints of the consortium agreement. The project consortium is bound to the EC by the project Description of Work (DoW), which states the project goals, its work plan and scheduling, the resources committed to perform it, the dissemination and sustainability plans for the results of the project, and some other details about it.

This document is a living one, expected to evolve as contributions are received. Should you be interested in contributing fixes, changes, new topics or any other kind of improvement, please contact the author at jgb @ gsyc.es.

2 What is FLOSS?

In this document, the term “FLOSS” (free, libre, open source software) will be used to refer both to “free software”, as defined by the Free Software Definition by the Free Software Foundation, and to “open source software”, as defined by the “Open Source Definition” by the Open Source Initiative. The term “libre software” is in some cases used as such (in English, although it is an Spanish and French word) to avoid the ambiguity of “free”, which can refer both to freedom or to cost (gratis).

Different actors may prefer any of these terms, and there are even whole communities that identify themselves as a part of the “free software movement” or “open source community”. In fact, for some people, the term used is quite important. However, the already mentioned definitions of “free software” and “open source software” have very similar effects: almost all the software that is considered “free software” by the Free Software Foundation is also considered “open source software” by the Open Source Initiative, and vice-versa. Some small fraction is not, but in most practical cases that fraction is so marginal that can be neglected.

FLOSS is defined by the actions that somebody who receives a piece of software can perform on it. But the current legal framework forbids almost all actions except that the copyright holder gives explicit permission for them. Therefore, the FLOSS status of a program is determined by the permissions that the copyright holder grants, which are usually encoded in the distribution license. FLOSS definitions mainly determine which licenses can be considered to grant the actions that they consider minimum in the case of free software or open source software.

As a conclusion, determining that some software is FLOSS software is not something arbitrary. To be considered as such by the wide FLOSS community, the licensing and distribution schemas used have to be recognized as such by the relevant bodies (the Free Software Foundation and the Open Source Initiative). To avoid confusion and criticism (and even some laughs) by the FLOSS community, a project shouldn't state that it is delivering "free software", "open source software" or "libre software" if that is not clearly the case.

Recommendation:

Ensure that all partners understand what FLOSS means in the context of your project. It is important as well that they realize which of the software they will be using or producing in the context of the project is FLOSS.

Recommendation:

When the DoW uses the terms "FLOSS", "free software", "open source software", "libre software" or any other related, ensure that they are properly defined, to avoid errors and miss-intepretetations.

3 Issues to deal with from the start

Even before a single line of code has been written, the project willing to produce some FLOSS should take some actions and decisions. It is important that they are considered as soon as possible in the project life, since many of them could shape the future of the FLOSS development process, or of its dissemination and impact. Early decisions may also help to avoid conflicts that could arise later, either between partners in the project, or with third parties, if no clear rules are established from the beginning.

However, some of them are moving targets, which could be reconsidered or redirected later, as the project evolves.

3.1 Determining FLOSS outcomes

Since the proposal stage, it is important to clearly determine the software components that are to be released as FLOSS. They can be, for example:

- Some components produced (or released) by the project. In this case, they should be specified as much as possible.
- All components produced by the project. This does not necessarily include previous work contributed by the partners or third party components.
- All components released by the project. This does not necessarily include third parties components, but includes all components released in the context of the project, including those that are derived works of prior works of the partners.
- The whole system used to deploy the demonstrations and test-beds carried on by the project. This includes all the components used to test the outcomes, including prior work components and derived works from them, and those obtained from third parties.

It is also important to determine how the FLOSS components are to be distributed. In the case of complete systems composed by FLOSS components, either the whole system can be distributed, or only the components separately.

Recommendation:

The Description of Work should specify clearly the outcomes of the project that are to be distributed as FLOSS.

3.2 Policies for licensing and transfer of copyright

Licensing is probably one of the most important topics that the project consortium should dealt with early. There are at least two reasons:

- Administrative. The intellectual and industrial policy of the consortium should be specified in the Consortium Agreement. This policy should at least allow for the release of the software produced by the project as FLOSS, should clarify the relationship with any other prior-art software that the partners may contribute to the project, and if possible, should state some common rules about copyright attribution and FLOSS licensing.
- Impact of the selected licenses. There are many FLOSS licenses, and each one could have a different impact on the relationship between partners, the business models for derived works, the collaboration with other FLOSS and/or R&D projects, the integration with other FLOSS and non-FLOSS components, and other issues.

It is important that the project selects the license or licenses to release the software having into account all these impacts. If the project consortium lacks the expertise to evaluate the foreseeable impact of FLOSS licenses, it would be convenient that they get some external advising on the matter. Given the complexity and novelty of software licensing in general and FLOSS in particular, and the different interactions with company intellectual propriety policies, industrial propriety, and others, this advising should be provided by an expert on FLOSS licensing.

Despite the convenience of this expert advising, some general rules that are usually considered as good practices are provided below:

- Ensure that the license is a recognized FLOSS license.
- Avoid license proliferation, at least if it is not rather well justified and understood. In other words, avoid producing a new license.
- Keep licensing as simple as possible. If possible, release everything under the same license.
- Consider the licenses of the FLOSS components that are planned to be integrated, so that they are compatible with the selected license.
- Consider dual licensing if needed for integrating and releasing FLOSS components with incompatible licenses.
- Ensure that the license selected is compatible with the envisioned business models, if any.
- If you plan to work with an already existing FLOSS community, discuss the license with them, or at least understand their licensing policies and views, to avoid future problems.

Remember that licenses used will shape future distribution of the produced software, and contributions to it. It has to be as easy to understand as possible for anyone considering using or creating derivative works of the programs. Therefore, using well known licenses, whenever possible, is a good choice: most potential users will be familiar at least with its main provisions.

It is also important to ensure that all consortium members understand the implications of the licensing schema chosen, and that they have checked that it is compatible with the intellectual propriety policies of their organization. In fact, in some cases, maybe some partners have policies that force them to use (or avoid) certain licenses. In some cases, this could lead to different partners distributing their components under different licenses. This should be avoided as much as possible, since it will make the licensing schema much more complex. But if the situation is unavoidable, at least license compatibility should be carefully checked.

Recommendation:

Avoid considering the decision on the FLOSS licensing policy as a marginal matter that can be dealt with at release time. Since the proposal is started to be shaped, it is important to consider the effect of the FLOSS policy on other areas of the project, such as development, dissemination, sustainability, and collaboration with other projects and communities. In cases where the project is producing outcomes expected to be the basis of a commercial service or product, the impact of the licenses on the business model should be carefully studied,

Recommendation:

The licenses to be used to distribute the FLOSS outcomes of the project should be specified in the Description of Work.

Recommendation:

Decide early in the life of the project who is going to be the copyright holder of the produced software. It can be the partner producing the software, the consortium, or any other third party (such as a FLOSS foundation, to which the copyright could be transfered). If possible, specify this in the Description of Work. Be sure of getting the needed legal advice about who can be the copyright owner given the structure of the involved institutions.

3.3 Previous assets and licensing status

In most cases, partners contribute with previous work, not covered by the provisions applied to software produced in the project. These cases can and should be clarified in the Consortium Agreement as much as possible. In addition, the licensing schemas for that previous work should be clarified, since it can have a great impact on the licensing choices for the results of the project. For example, if the goal is to produce a completely FLOSS system, having some partner with prior work, needed for the system, but which is only available through a non-FLOSS licences would render this goal impossible. That would be the case even if that work is available for gratis.

When those cases arise, it is important to remember that if the involved partners have total ownership over the code, they could relicense a version (or a derived work) under a FLOSS license. They can do this even while maintaining other versions of the same component under a non-FLOSS license.

In addition, license compatibility should also checked. Even if all the previous work is FLOSS, it could be the case that the result of integrating it cannot be distributed. This usually happens when the results are derived works of several components, and there is no way of simultaneously satisfying the conditions of the FLOSS licenses for each of them.

Recommendation:

As soon as it is possible, the partners should produce a list with all the prior work components they intend to use, with the licensing status. That listing should specify at least the licenses under which those components are distributed, and include links to where the software can be downloaded, so that other partners can check. Once the list is complete, the partners should check whether such list is compatible with their goals, in terms of FLOSS licenses for specific results. In case the check is negative, corrective actions should be taken. most of this work should be done when the project is still in the proposal stage, since some problems (such a partner refusing to distribute a key component under a FLOSS license) can be unsolvable at later stages.

Recommendation:

To include in the Description of Work the listing of prior work to be reused, its licensing status, and the corrective measures to be taken in case license compatibility arises at later stages. Since the Description of Work is binding for all the partners, this will help to avoid difficult situations in the future, if some partner failed to disclose the status of some prior work.

3.4 External dependencies

Usually, projects use components from third parties as a part of the outcomes, or of the platforms needed to run or test the outcomes. The dependency on a given component can be clear and explicit, or in some cases be hidden under a complex chain of dependencies from other components.

The licensing schemas of all those components coming from sources outside the consortium could interfere with the FLOSS policy of the project. Therefore, it is important to carefully check the licensing status of all external components, to decide whether their use (or, if needed, creation of a derived work) will allow to distribute the final outcomes under the FLOSS licenses decided.

Recommendation:

To perform, as soon as possible, a dependency analysis to determine all third party components needed for the outcomes of the project. Analyze the licensing status of all of them, checking for FLOSS status, license compatibility, and its impact on the FLOSS policy of the project. Whenever possible, this should be done as soon as possible, even in the proposal stage. If possible, clarify all these matters before the first version of the Description of Work is complete, and include in it the resulting list and status of third party components.

3.5 Development model

FLOSS can be developed in many ways. Traditional, in-house methods can be used, but other, more community-based practices can be worth exploring. Since partners in the consortium are expected to collaborate, and in some cases to do some joint development, community-based practices can be specially interesting.

Using open, community-based development practices may have several advantages and little risk for projects deciding to produce FLOSS components. These practices usually mean to develop as most FLOSS communities do, with most (if not all) technical decisions taken after public discussions, with living public repositories for source code and issue tracking and planing, open communication channels between developers, and open interaction with users.

For an R&D project which has decided to produce FLOSS, the some of the specific advantages of this model are:

- It is easier to involve external contributors, since they have access to all the relevant information. This can have a huge impact in the sustainability of the project after funding is finished.
- Third parties can provide relevant feedback, helping in the testing and validation phases.
- This practices match well the traditional uses of the scientific community, based on the open discussion of all the aspects related to the research work.
- There is an extra of visibility, since people interested in the technology will probably look at the project for information. With time, it can become the most visible point for that information.

- Confidence on the results of the project can be much higher if there is early access to all the information related to it.

Maybe the worst risk to be faced is that external experts could examine early the results of the project, or the methods to reach them, and criticize them. But even in this case, that criticism could be also valuable feedback, which could help the project to re-focus or consider issues that had not been detected. In addition, the pressure that can mean for project partner to be subject to public scrutiny may also help to achieve excellence, and will certainly help in the accountability of the project.

Recommendation:

Include in the Description of Work the development practices to be followed for FLOSS components. Consider being as open and as much "community-based" as possible, as a way of improving the results and impacts of the project. Have as a model the working practices of community-based FLOSS projects.

3.6 Relationship with external parties (users, contributors)

When FLOSS is produced, the usual barriers between developers and "others"blurs. External contributors can provide valuable patches fixing errors or providing new functionality. Many users are willing to contribute with bug reports, feature requests or casual support to other users. Any of them can contribute with wise ideas in many technical and non-technical discussions. Therefore, taking advantage of the actions of all of these actors should be important for the project. In addition, in many cases the creation of a community around the FLOSS components produced is an explicit goal of the project.

But maintaining good relationships with those external actors is something that cannot be taken for granted. They can feel excluded, their contributions not being taken into account, if they cannot even find ways of contributing or getting involved in the project.

Therefore, an specific policy and means for maintaining a healthy relationship with external actors has to be put in place.

Recommendation:

Include in the Description of Work a carefully designed policy for managing the relationship with external actors which could contribute to the project.

3.7 Envisioned sustainability model

Once the project ends, the FLOSS components remain available, as long as they can be downloaded from somewhere in the Internet. However, this is not enough for those components to survive: they need to adapt to new circumstances, and to evolve and improve. Even the most interesting and functional components become irrelevant if they are not properly maintained.

Acknowledging this, and also having into account that the R&D project is usually a first step towards the marketing of some technology, product or service, most projects look for ways of ensuring the maintenance and evolution of the components once the funding is over. However, not in all cases this issue is addressed with the detail it deserves. For instance, merely stating that the users and developers community will take charge of maintaining the software after the end of the funded project is not enough: the community has to be created, nurtured and maintained.

In addition, not always the creation of a community is easy or even feasible. If the potential users of the components are a small population, or if the component is not mature enough to attract real users, or if there is not enough economic incentives for companies to join, or if nobody is contributing with the resources to bootstrap a community, it will not happen.

Therefore, it is important to specify which sustainability model is to be used after the funded project ends. If it is community-based, how that community will be created and maintained should be detailed, including which resources will be allocated for that.

Of curse, some of these issues are difficult to foresee when the project is planed. For example, in many cases it is difficult to estimate how many users or external contributors could be attracted to the project, which means that plans for raising and maintaining the community (or in general, the sustainability after the funded period) will be mainly guidelines, that should be adapted during the life of the project.

Recommendation:

The Description of Work should include a clear plan for sustainability after the end of the funded period. If it is based in the maintenance by a community, the work plan should specify how this community is to be raised and maintained, and the resources allocated to that end. In addition, it should be explained how this community is expected to find and organize the resources needed to maintain the component after the end of the funded period. If some other sustainability model is to be explored (public or private sponsorship, business agreements, business models, etc.) they should also be explained to some extent.

4 Development issues

If the project has decided to produce FLOSS components, there are some issues related to the development process that should be taken into account. In short, most of them relate to how efficiently they can get advantage of the practices and resources that FLOSS development communities usually employ. In addition, the project should also do things in the way that other FLOSS developers expect, if it expects to attract their attention, with the idea of forming a community.

4.1 Infrastructure for supporting development

Over the years, FLOSS projects have developed an elaborate set of systems and practices for supporting the distributed development models they usually employ. Most of them could also be suitable for the development carried out by R&D projects (usually carried out by developers from different partners, geographically dispersed), and even for other project coordination tasks.

The basic facility used by FLOSS developers is the forge. A forge is a site providing a set of facilities to support software development, which are managed (to a certain extent) in coordination. Some of the services usually provided by a forge are (the list is not exhaustive):

- Source code management system. They are used to maintain all revisions of the source code. There are two main types of them: the centralized ones (CVS, Subversion) and the newer, distributed ones (git, Bazaar or Mercurial).
- Release repositories. Used to maintain releases of the code or documentation produced by the project, normally ready for users.
- Issue tracking systems. Used to keep track of bug reports, feature requests, and other kinds of tickets.
- Mailing lists. Used for asynchronous communication between members of the development and users community.
- Forums and blogs. Used for public information of news, events and comments related to the project.

- Wikies and content management systems. Used to maintain documentation, or other information about the project.

In addition to some proprietary software, there are several FLOSS products providing all or a part of the forge services. The most common are GForge and its derivatives (including Fusionforge), Trac, RedMine, Plone Forge, Gitorious. Those can be deployed by a project to provide the forge services they may need.

However, a project does not need to deploy its own forge. Usually, it is more convenient to use the services provided by a third party. In addition to companies providing forge services for a cost, there are several forges that offer their services gratis to FLOSS projects:

- Public forges, such as SourceForge, Google Code, Savannah-nonGNU, Berlios, Github or Gitorious. Any project developing FLOSS can be hosted in them. Usually, all the information maintained in it (except for privacy-related information) will be public.
- Project forges, such as Savannah-GNU, Alioth, or the Apache, GNOME, Mozilla or KDE development sites. They are maintained by a FLOSS project for hosting their own development.
- Community forges, such as OW2 or Morfeo. They are maintained by FLOSS development communities, in which companies have a prominent role, for the software developed by them.
- Specialized forges, such as OSOR (for European public administrations).

All this wealth of facilities is at the disposal of disposal of R&D projects producing FLOSS. They should explore how better benefit from them, and to which extent use them to come closer to their goals. This could include the raising of a community (attracting external contributors), the dissemination of the produced software, etc.

Recommendation:

Projects should describe in the Description of Work whether they are planning to use the services of a forge, and if so, how. It is important to notice that they can use these services not only for software development, but also for project coordination and production of deliverables. For instance, most projects need mailing lists for coordination: those can be easily deployed using forge services. Repositories for documents, and source code management systems may also be of help in the elaboration of materials, including written deliverables.

Recommendation:

Projects should decide early in the life of the project, and certainly before any development starts, which specific forge to use (or deploy, if they want to have their own). They should also detail their policy with respect to the use of the forge: which services they are going to use, which kind of information should go trough them, mechanisms used for getting external feedback and communication with third parties, etc. That forge policy should be specified as a part of the early design and detailed planning deliverables to be produced at the starting of the project.

4.2 Development tools and practices

In addition to using the services provided by a forge, the developers in the project should be familiar with the common development tools in the relevant FLOSS areas. Each kind of FLOSS development tends to have a certain set of tools and procedures that are “expected” by other developers: not using them means erecting barriers to receiving contributions, and making use more difficult. However, those tools and practices may change a lot from area to area. They range from the existence of certain files in the source code (such as the COPYING, README or CHANGELOG files) to the use of certain configuration and building tools (such as Makefile, autotools, Ant or Maven), or even certain Integrated Development Environments (such as Eclipse).

It is important that the developers in the project know about these practices and tools, get familiar with them (if they were not), and honor the common uses of the FLOSS area in which they are developing, to a reasonable extent, if they do not want to be perceived as aliens by other developers in that area.

Recommendation:

Projects should ensure that they include at least some expert FLOSS developers in the area they are working. They should introduce the rest of the developers to the common uses in that area, and help them to adapt those. It is important that they explain how other developers expect the source code and related information to be. If needed, they should produce some guidelines about those matters. Except when other reasons may conflict with this, it would be important that developers in the project use the same tools that other FLOSS developers in that area. That would help a lot when receiving external contributions, or when sharing code with them.

4.3 Release early, release often

One of the main recommendations of Eric Raymond in its seminal document about FLOSS development, “The Cathedral and the Bazaar” was to release early and to release often. This is of course still valid, and should be considered of extreme importance by research projects. In addition to the reasons that Eric Raymond exposes, there are some others, maybe specific to research projects:

- The live of the project is relatively short, usually from two to three years, in some cases four years. That means that if the project waits to release until it has a mature product, or even an usable product, probably the project will end before anyone external to the project has had the time to know enough about it to contribute back. This will render many of the dissemination efforts of the project irrelevant, specially if they are related to the raising of a community.
- Technology evolves fast. Therefore, the window for showing results that are ahead of the state of the art is small. The sooner the project releases, the most likely that the window is still open.
- Releasing early and often means that there are no hidden agendas: everything happens in the open. This will be perceived by other FLOSS developers in the area as a sign of maturity and trustability. In addition, seeing frequent releases tends to be a good indicator of a healthy development effort.

In fact, probably there are little reasons not to have a policy of “continuous release”, which means that the development is carried out directly in the source code management system (SCM). In this case, usually a “stable” branch is defined in the SCM, to which external developers are pointed. That branch is maintained with special care, trying to have it always ready, at least for testing. Of course, in addition, some regular releases could be done in cycles, freezing functionality for a while, and after performing the convenient testing.

Recommendation:

Define a release policy in the Description of Work. If possible, use development methodologies (such as spiral development) compatible with continuous or semi-continuous release. Start the releasing of source code as soon as possible: from the very beginning in the case of code in the SCM, and as soon as there is a semi-usable (even with very minimal functionality) version.

4.4 Integration with established communities

One of the most effective ways of ensuring the sustainability of the FLOSS components after the end of the funded period is by integrating it in an established community. This will help to find other interested developers in the components, as well as to gain visibility of them.

In this respect, there are at least two chances:

- Approach a domain-specific FLOSS community to which the project could contribute with components important to them. For instance, if the project is producing elements missing in (and interesting for) KDE, GNOME, Mozilla, Eclipse, Apache, Debian, etc., those communities can be approached with the intention of joining them. Usually, they have procedures for accepting external contributions such as those produced by the project, or even for hosting the development (such as the Apache Incubator).
- Approach a company-based community, such as OW2 or Morfeo, which could also be interested in the outcomes of the project.

Recommendation:

Consider as soon as possible, even at the proposal stage, to identify communities that could be interested in the outcomes of the project. In some cases, they could be approached with the intention of adding them to the partnership of the project. In others, they could be approached with the intention to join them. In any case, that should be specified as a target in the Description of Work, with the actions and resources needed to reach that end. In addition, it would be convenient to specify alternative plans, because no matter how much effort is devoted to involve one of those communities, the success depends on the relationship between both parties, and in how other developers in the target community may perceive the project outcomes.

4.5 Openness of procedures

Although FLOSS does not mandate that other information, apart from the source code itself, is made public, having a general "open" policy in other areas may have useful results. Public accountability of what the project is doing and how it is doing may, for example, help to create the trustworthiness needed to attract external users, and later contributors. Exposing information about detected bugs, roadmaps, design decisions, etc. help to create the kind of atmosphere in which FLOSS developers tend to find themselves more comfortable.

Recommendation:

Try to keep as much public information about the project as possible. This could include all non-confidential information, including minutes and slides of project meetings, design documents, research papers, and of course all development-related information. Make it publicly available in a prominent area in the project website.

5 Distribution and dissemination

One of the main advantages of creating FLOSS components is that they can be redistributed easily, which should help to disseminate the results of the project. It is therefore important to make the most of this advantage, by taking measures to boost distribution of the produced software and other results.

5.1 Software distribution policies

The project has to decide on the policy for the distribution of the FLOSS it releases. Usually, not only source code should be redistributed: binary and packaged versions, ready to install and run, are a good complement, which may lower meaningfully the barriers for adoption. In general, packaging for the target platforms (eg., RPM and Debian packages in the case of Linux-based platforms, or Windows-autoinstallable packages for Windows platforms) is the preferred form of distribution, since users have less trouble with those formats.

The FLOSS components released by the project can be distributed separately. But whole systems, ready to use, can be produced as well, and offered for download and redistribution. Prominent cases of whole systems include custom Linux-based distributions, ready to install in a USB memory stick configured to bootstrap a computer, and virtualized images, ready to be deployed in a cloud.

The project should also decide whether each partner will redistribute their components, or if all of them will be available in a single location. As a general rule, the latter is preferable and easier to maintain, since it contributes to reinforcing the image of the project as a whole. However, both distribution policies are possible simultaneously: FLOSS components can be distributed from different locations.

In many cases, the software can be distributed as a part of a collection put together by a third party. One of the most relevant cases is Linux-based distributions: if the software enters for example Debian or Ubuntu, its visibility is hugely augmented. However, this only happens when it has a clear interest to end users, and attracts the attention of the developers of the distribution. The most sensible action to help this happen is to provide proper packaging for those distributions, since that simplifies the work of distribution developers, if they decide the component is worth considering.

Recommendation:

The project should decide, and express it in the Description of Work, its policy and plan for FLOSS distribution. Whenever possible, a central repository with both source code and binary packages ready to install should be established, maybe using the services of a forge. The project should devote the necessary effort to maintain the installable packages as much updated as possible, and should establish clear channels for receiving feedback from users installing them.

5.2 Contributions to external projects

It is common that the project uses external FLOSS components. In this case, some patches fixing bugs or adding improved functionality could be produced as a part of the activities of the project. In this case, it is usually in the benefit of the project to contribute back with those patches to the FLOSS projects producing the components. That way, there are chances that the patches are integrated in new releases, saving the project the burden of re-patching them.

Recommendation:

Establish a clear policy, detailed in the Description of Work, to contribute back fixes and improvements to FLOSS components used by the project.

5.3 Dissemination strategy

It is usually recognized in R&D projects the importance of designing and implementing a good dissemination strategy. In the case of FLOSS components, this is of course also the case. Moreover, FLOSS has its own places, channels and procedures for dissemination, which can be exploited in addition to the more traditional ones.

5.4 Conferences and meetings suitable for dissemination

The wide FLOSS community has several meeting points, quite specific for FLOSS-related issues. Some of the most relevant of those meetings in the European Union are:

- FOSDEM. Targeted mainly at worldwide FLOSS developers. Organized in Brussels (Belgium). A very good place to show applications to FLOSS developers.
- LinuxTag. Targeted also to developers, but mainly to companies. Organized in Germany (last years it was in Berlin). A good place to contact business, includes the largest expo about FLOSS in Europe.
- Open World Forum. Targeted mainly to business, but with a wide focus. Organized in Paris. Good place to show ideas to a large community, and to organize workshops for relationship with other projects.

There are also plenty of national or regional conferences (some of them partially running in English, and accepting international presentations), and project-specific meetings, such as GUADEC (for GNOME), Mozilla Drumbeat (organized by the Mozilla Foundation), KDE Academy (for KDE), etc. All of them could be suitable places for showing the results of a project if they could be of interest to those communities.

Recommendation:

If the FLOSS community has been identified as a dissemination target by the project, an specific plan considering some of these conferences and meetings should be designed. It could be a part of the Description of Work, or of the deliverable detailing the dissemination strategy and plans.

5.5 Other communication channels

There are many other, usually web-based, channels suitable for communication and dissemination of the results of the project to parties interested in FLOSS. Some of them:

- The forges themselves (and specially the largest ones) may be a dissemination channel, since they are used for searches for specific software. In this case, the correct categorization of the project is of paramount importance. In some cases, a fake project (including only categorization information and release versions for download) can be open, even if the actual development takes place elsewhere.
- Freshmeat. This is a website specifically designed for announcing FLOSS products, and their new versions.
- Ohloh. This site is a website which provides analysis of FLOSS projects and products. New projects can be registered.
- Identi.ca, Twitter groups and tags.

Recommendation:

Include some (or all) of these channels in the dissemination policy.

5.6 Demonstration site and materials

When approaching FLOSS developers, and in general, almost anyone in the FLOSS community, the principle of "show me the code" (or "show me the binary", for that matter) has to be taken into account. In other words, people is used to quickly test the program, if they think it could be interesting for them. Therefore, any dissemination action has to be complemented with a version of the produced software suitable for use and inspection: otherwise, the project will probably be tagged as "vaporware".

In addition to the binaries, when possible demonstrations sites and materials showing how the components behave (such as installation and user manuals, screencasts or other kind of videos) should also be produced, and included prominently in the project website.

Recommendation:

The project should design the development process in such as way that a preliminary version of the FLOSS components is available early in the project life, and certainly at about the same time when the dissemination effort starts. In addition to the released FLOSS components, other materials helpful for their users should be produced.

6 Community, sustainability

An R&D project producing FLOSS components usually has as a goal the raising of a community around them, that could help to (or directly, ensure) the sustainability of the components after the end of the funded period.

6.1 Growing a community

Many R&D projects have as a goal the rising of a community of users and contributors that help in the sustainability of the project once the funded period is over. However, the rise of such a community cannot be taken for granted. On the contrary, specific plans and resources have to be designed and devoted to this end.

Recommendation:

If the project has the intention of growing a community around the FLOSS components it will produce, include in the Description of Work an specific plan and the needed resources to accomplish that task.

6.2 Passing the token

When the funded period for the R&D project ends, different partners in the consortium may have different strategies with respect to the produced FLOSS components. Some of them may move on to new research challenges, some others may still use the components, but will have not enough resources or interest to collaborate in their maintenance, maybe some others will still maintain them.

Therefore it is important to define who will have, at the end of the funded period, resources and interest to carry on the maintenance and further development of the components. If no partner will play that role, the project should design an strategy to pass the token to other interested parties, if any. This is something that should be done during funded period, so that some resources can be devoted to the transition.

Recommendation:

The Description of Work, as a part of the sustainability plan, should specify which partners will be responsible for the maintenance of the FLOSS components after the end of the funded period. If no partner can be specified, some transition strategy should be designed, identifying the third parties that could be interested in getting that responsibility. In some cases, some FLOSS-related foundation could be interested in the future maintenance of the project.

Recommendation:

In any case, the project should ensure that at least the FLOSS components and their accompanying materials (such as their documentation) will remain available well after the end of the funding project. This can be done by uploading the software and materials to some forge or website that ensures their long term availability.

7 Other outcomes of the project

7.1 Licenses for public documents and media

Public deliverables produced by the project could be free documents (as defined by the "Definition of Free Cultural Works"). Probably they should, in any case, if they are needed or convenient for understanding the released FLOSS products. Although the definitions of FLOSS do not mandate this, the project may experience little harm because of it. On the contrary, having the documents distributed under free licenses will ensure that they can evolve and be redistributed with the FLOSS components that have relationship with them. In addition, this usually helps to dissemination of the documents, since third parties can easily participate in the distribution chain.

Recommendation:

Consider including in the Description of Work a default policy for releasing all public deliverables as free documents. In this case, make the consortium familiar with the definition of "Free Cultural Works", and specify also a license for distribution, as soon as possible in the life of the project. The Creative Commons Attribution-ShareAlike license is one of the most commonly used by projects in this situation.

7.2 Formats for documents and media

When interacting with the FLOSS community at large, usually some care has to be put on the formats for the documents and media used. First of all, the project has to understand that many people in that community will refuse to use non-FLOSS tools, or will not have the chance of doing so. Therefore, avoid as much as possible formats that cannot be viewed with FLOSS tools. If the document or media is intended for edition or change, ensure that can be done with FLOSS tools too.

If the project wants to project an image of being FLOSS-friendly, it is also important that uses formats usual in FLOSS communities. For instance, ODF is usually preferred to other formats for editable documents, although PDF is acceptable for viewable-only documents. Video and audio formats based on Ogg are preferred to those based on MPEG or Flash.

If the partnership includes FLOSS organizations, or other partners heavily involved in the FLOSS world, it is likely that they will propose these formats to be used also by the project for its internal use. To avoid problems once the project is running, these aspects should be clarified as soon as possible, even when in the proposal stage.

Recommendation:

Consider including a policy about formats for documents and media, both for public and internal use, in the Description of Work of the project. Try to make this policy as much friendly as possible to potential users and contributors, and to the FLOSS community at large.

7.3 Software used in dissemination and development activities

If the project wants to show some commitment to the FLOSS community at large, it should be as careful as possible with the software used, at least in its "visible" activities. Among them, those most scrutinized could be dissemination and development. Tools accessible by third parties in those domains should be FLOSS themselves, to a reasonable extent.

Recommendation:

If the projects wants to be "FLOSS-friendly", it should ensure that it will use FLOSS components for its internal and external activities, as much as possible. A strategy and some criteria for determining which components will be used, and which exceptions will be allowed could be a part of the Description of Work.

8 Other issues

Transparency and accountability will in general be appreciated by the FLOSS community at large, and will easy the collaboration with third parties. That will also help others to understand the goals of the project by potential contributors and users.

Therefore, it is important that public deliverables are really public, and easily accessible from the website. For the same reasons, a version of the Description of Work, maybe stripped out from its confidential information, should be published and easily accessible.

Recommendation:

Specify in the Description of Work whether there will be public version of it, and how will all the public versions of the deliverables be made public. Ensure, as much as possible, that they remain available after the end of the project.

9 Further reading

[To be completed]