Why and how to contribute to libre software when you integrate them into an in-house application ?

Christian Bac, Olivier Berger
GET / INT / Département INF
Évry, France

Véronique Deborde, Benoît Hamet
GET / Direction Scientifique
Paris, France

`<christian.bac@int-evry.fr>`, `<olivier.berger@int-evry.fr>`,

`<veronique.deborde@get-telecom.fr>`, `<benoit.hamet@laposte.net>`

Abstract—Free or open source software are common tools that everybody can use and customise at its convenience to create in-house applications. Using and customising free software is not sufficient to ensure that this in-house application will be maintainable at mid or long term. This paper draws lessons from our in-house project, the development of a groupware Web platform for researchers, to help defining a policy through which efficient contributions can be made to open source software so that the in-house projects may remain viable.

Index Terms—efficient contribution to open source software, in-house applications, maintainability, tools for researchers, collaborative work, wiki, Web interface, software engineering, free/libre software, phpGroupware

I. INTRODUCTION

Libre software[1] is seen today as a wonderful opportunity to reduce the duration and costs of the software projects, and increase the quality of the applications. A very large number of software components or high-level applications are available and ready to be integrated, customised, in order to create applications that fit the needs of organisations.

Hopefully, it is not necessary to participate directly, or even indirectly to the development projects of these libre software components, to be able to use them. The development of an initial version of such an internal application, based on external pieces of free software, may require the modification of these components and the development of some "glue". These tasks may be done in a very classical way for a software project[2]. The fact that the components are libre software change very few aspects of the project's management, in the end, provided of course that the integrated components are operational and deliver what's described in their specifications[3].

Beyond the initial integration phase of the life cycle of a new in-house application, comes the issue of the *mid-term maintainability*, especially when time comes to react to the evolution of the external components that have been integrated. Of course, software engineering laws tell us that the earlier organisations prepare plans to address potential future risks, the lower the consequences will be, should any hazard occur to the project.

We will show, through the experience gained in developing our Web platform for collaborative work *ProGet*, that it is necessary to define a policy which states how the internal team can get involved in some external development projects of those libre software pieces that are integrated into the application.

Through our participation, we can ensure that these external projects will be able to integrate into their code the evolutions that have become necessary to the needs or our particular integration, and that they will carry out, in the future, the maintenance of these new elements. This is the best way that some part of the internal application maintenance efforts can be *outsourced* to the community, and thus reduce the costs for the internal teams.

This paper is organised as follows: the next section presents research in our institution and our project's goals, section III describes some previous work done with free software, section IV details the different free software that we integrated to create this project, section V details some lessons learned during the project, and the last section concludes and draws some perspectives.

II. PROGET AND RESEARCH AT GET

The GROUPE DES ÉCOLES DES TÉLÉCOMMUNICATIONS[4] (GET) is composed of several engineering and business schools together with research centres in Paris (ENST), Brest (ENST Bretagne) and Évry (INT), in France. Highly-skilled students are selected by competitive examination and will be awarded a master's degree

---

[1] In this paper, we assume the following wording relate to the same concept : libre software, free software, open-source software.

[2] Although this is sometimes said about libre software, note that it is not true that one is forced to publish modifications made on some piece of external free software, be it published under the GNU GPL [1], as long as the modifications are kept internal to the organisation.

[3] There is often no warranty by a vendor on the conformity of the software to its specification or documentation, but that's also the case for many proprietary software elements, and a preliminary step to any software projects is to test the components used for the technical foundations.

[4] `http://www.get-telecom.fr/`

Proceedings of the First International Conference on Open Source Systems
Genova, 11th-15th July 2005
Marco Scotto and Giancarlo Succi (Eds.), pp. 113-118

after three years of intensive study. We also have some hundred new PhD students every year.

GET represents a pole of reference integral to the French public research structure in the field of Information and Communication Science and Technology. The research teams are made up of more than 600 full-time research equivalents.

The range of the researchers' expertise, from technologies to social sciences, enables the integrated approach so characteristic of GET research and fosters its adaptability to new application sectors and new usages in response to current challenges in the field of Information and Communication.

To give a clearer view of research at GET, the research office started to catalogue the activities from the different locations so that the research may be described in terms of research *projects* and *programmes*. A project is made-up of a group of people working together on closely related subjects, for example we belong to the "Collaborative Platforms for Research" project. A programme associates different projects loosely related; for example our project is related to the "Web and Information society" programme. Due to the fact that GET teams are located in different areas, the research office also decided to propose a Web platform to help researchers collaborate through groupware tools and animate their research work.

A. Goals of the ProGet collaborative platform

ProGet has been launched in July 2003 with the following three goals:

1. provide research teams in GET with cutting-edge technologies in terms of Web based *groupware* tools;
2. allow the research office to manage the *administrative* informations about research projects such as the project description, the list of people working in the project, the yearly and mid-term goals, etc.;
3. generate a public *Web portal* based on information extracted both from the administrative part of the system and the groupware.

Those latter two are quite specific to GET needs, so the next subsection focuses only on the generic groupware tools for researchers in higher education institutions, which should be of interest for most readers.

B. Groupware tools for researchers in higher education

The following features have been selected as the basic needs for collaborative work in the context of research activities at GET:

Document sharing: people in the same project must have a way to share the documents they produce, whatever the type of document.

Asynchronous communication: each project needs predefined *mailing-lists* and also capabilities to create mailing-lists at its needs.

Online editing: people in a group are allowed to edit pages using their Web browser to create a collab-

orative Web in the *wiki* way [2].

Publishing short announcements: the project manager can write short news items about the project, and have them published on-line on the public Web portal.

As a postulate, the system should be designed as a Web platform to allow people to access it without the need to install any specific application on their computer, and from wherever they have to work (potentially anywhere on the Internet).

The system must provide a Web interface accessible with any browser, and also allow people to interact with the platform with other tools of their choice, for instance DAV-compatible file managers, to share documents with colleagues or publish information about their projects in very simple ways (and of course, be they using Microsoft Windows™, or GNU/Linux systems).

Especially, information generated in the collaborative part of the platform should be made accessible to the public portal without further interaction with a webmaster.

III. Previous work with libre software

As described in [3], we and other GET researchers and students, contributed in the development of a collaborative platform called *PicoLibre* [4]. This platform is targeted at software development, to help students in Computer Science curricula and researchers to develop and host their software projects, by offering the necessary collaboration tools (mailing-lists, CVS repository, issue trackers, etc.) in a similar way to SourceForge [5]. Another goal of the PicoLibre project was to foster the publication of projects as free software, since we could introduce the users to the practice of the common tools used everyday by libre software developers.

This platform was created as a free software tool (published under the GNU GPL license [1]), using free software components (some already existing, like phpGroupware or Sympa, and others developed for PicoLibre). Several PicoLibre instances are in operation since fall 2001 in GET sites, and outside.

In some aspects, PicoLibre is very successful, since it allows us to host a large number or software projects with a minimum burden for its administrators. It is also sometimes used for projects not specifically targeted at software development, for teams that need a collaboration space and associated groupware tools.

The PicoLibre project has given us a good amount of knowledge on the advantages and constraints of the design and development, but also the administration, of a collaborative Web-based platform in the context of a higher education institution.

PicoLibre failed to certain extents, especially with respect to its maintainability, and in its capacity to be integrated to a libre software distribution. The project was done in so a short time that developers only concentrated on the innovative parts and spent too little effort to keep the software developped in the main stream of

Proceedings of the First International Conference on Open Source Systems
Genova, 11th-15th July 2005
Marco Scotto and Giancarlo Succi (Eds.), pp. 113-118

the free projects it was related to (phpGroupware [6], for instance).

Although having been used intensively by teams of researchers or students investigating software development, PicoLibre wasn't well suited for generic needs of teams of non software developers. For example the CVS [7] source code repository is a central tool in PicoLibre. It is very useful for software development, but doesn't fit well for casual researchers uses. For example, although it is possible to manage revisions of a document written with an office suite in CVS, it is not very convenient, since it requires installation of a specific CVS client program. Most non software developers (for instance researchers in the field of finance or business administration) are thus not comfortable in using such a tool.

Considering the aforementioned thoughts, we proposed to start and build the ProGet platform using some of the free software modules that had been used to create PicoLibre, and to combine these modules with other existing libre software projects to fulfil new capabilities. We intended also to develop this platform in a much more maintainable way.

IV. Integrated libre software components

Installed on a GNU/Linux operating system and dedicated hardware, the ProGet platform is composed of various software components. We will not describe every aspects of this system[5] to concentrate on the libre software high-level applications that have been integrated or enhanced in order to complete the platform.

Many other essential libre software tools are used in addition to those described bellow, among which some like OpenLDAP [8], for instance, which forms the heart of the authentication and identification, and which is essential to "glue" all the high-level applications together, by sharing some common user data model.

A. Features of the integrated libre software high-level applications

The Web platform consists in several applications; all these applications are free software tools that we glued together to form a complete collaborative platform. The applications are :

Apache Web server: [9] the classical Web server on which the applications run, and that serves Web pages for the projects

Documents repository: the Apache WebDAV [10] server;

phpGroupware applications: the phpGroupware [6] applications server, serving its classical groupware tools, the public portal, and the management application for the research office;

Sympa mailing-list manager: [11] the powerful mailing-list manager for the projects' communica-

tion lists;

Twiki Wiki-Web server: [12] the powerful wiki (Web based "white-board" system) providing a Knowledge Base tool for the projects.

All these tools are themselves among the best free software solutions available in each field. Instead of trying to add to one of these tools the missing functionalities found in others, it seemed more efficient to try and glue those different tools together, although there may be some redundancy in their functionalities.

B. Modification and customisation efforts

Most of our software development effort has been targeted on the following applications (by order of magnitude):

phpGroupware custom components for management of research projects and Web portal: These developments were very specific to the needs of the ProGet platform, and won't be published outside GET.

phpGroupware core components : among other minor enhancements and bug fixes, we have improved the interaction mechanisms between phpGroupware and a LDAP directory, and the internal phpGroupware virtual file system (VFS) storage modules, in order to use a WebDAV repository. These needs are quite generic and not really specific to the ProGet platform, and greatly improve phpGroupware as an advanced groupware environment. These modifications have been contributed to phpGroupware (see subsection V-C).

Various bug fixes and modifications to the other components : many small improvements or bug fixes, and some specific modifications have been made on the DAV server and LDAP interface of Apache, or on the *Twiki* and *Sympa* programs, mainly to facilitate the integration of all these components together with OpenLDAP, or in order to structure the data-models or interfaces of these applications around the core *research project* entity. Some of these software developments were contributed (as patches for instance), and some kept internal because they were very specific to ProGet.

In the end, although our software development effort was substential for an integration project, it was far lower that what would have been required for the in-house development *from-scratch* of a brand new system. Taking advantage of existing high-level libre software applications was the only way to attain the broad spectrum of features we needed, with a budget of only of dozen man-month to start of. The developed platform provides generic tools much more interesting for the casual GET resercher, compared to PicoLibre, as well as the most advanced functionnalities demanded by expert users (DAV, Wiki, etc.). Year 2005 will see the official launch of the ProGET platform for all GET users.

---

[5] The authors plan to publish a more detailed description of the system, for a reference which would describe technical characteristics of the tools, challenges, and issues solved, but which are not really relevant in the perspective of the present paper.

Proceedings of the First International Conference on Open Source Systems
Genova, 11th-15th July 2005
Marco Scotto and Giancarlo Succi (Eds.), pp. 113-118

V. Lessons learned

This section draws some lessons learned during the project. One of our first tasks was the definition of a policy to adopt with respect to our contributions to external libre software projects. The problematics with defining such a policy is developped in subsection V-A.

After a substential developement effort collaboration on different libre components we are also able to clarify our vision of free software projects' quality "myths", as explained in subsection V-B. The subsection V-C describes our contributions to one of the key *upstream* projects used in ProGet, and finally, the last subsection deals with some limitations of our contribution efforts.

A. A policy for contribution to libre software projects

Our internal project is deeply dependant on others' libre projects, that we call *upstream* projects. It was then important to define a clear policy on how to contribute to the upstream projects we would have to use during the development. As described in subsection V-A.1, there are some challenges to face in defining such a policy.

In all cases, the new project is vulnerable to the hazards of upstream projects as we will see in subsection V-A.2.

Cooperation with external bodies has to follow some specific path. According to social rules of free software projects, it is necessary to contribute to some extent to be able to take advantage to the cooperation with the upstream projects. As described in subsection V-A.3, this collaboration is possible once a new wannabe contributor has gone through the developer integration process.

A.1 Challenges

There are at least two ways to build a platform such as ProGET.

The first one is by making it in the *quick and dirty* way. This way was more or less the one used in the PicoLibre project. The method is simple: taking a "snapshot" of each free software needed, then build your code with this basis. At the end, releasing your code (internally or to the public). If the project is managed in this way, it has nearly no real impact on external projects. But as the final application depends on particular versions of libre components (those present at the time the snapshot was taken), its code is hard to maintain because the external modules which have been integrated "live" independantly. They will have to be "patched" again, each time a new version is released by the upstream projects, to be integrated back into the application.

The main advantage of this method is that it reduces the initial development costs. But the main drawback is that it is necessary to do lots of code maintenance each time the internal project needs to take advantages of new external releases of the underlying free components, for instance when new features are interesting, or

whenever security fixes are issued. Besides the burden for maintaining the in-house solution, it is also a bad idea to have this approach when intending to publish this new application's code "as-is" (for instance as free software), because it would place also the same burden on potential users of that new application. Although there is no formal responsability to them, they may suffer big inconveniences if deploying this unmaintainable application and angryness may be the only result gained.

The other way is more difficult and expensive. The project must adapt in order to contribute to the free "upstream" projects it depends on, and try to modify external code and tailor it so that it fits its needs. This means that patches on those components that prove necessary during the development should be integrated by the upstream projects. This method has the great advantage to guaranty that the in-house project will be easier to maintain (less patch sets to apply again and again) and that it is better integrated with the upstream projects, to adapt to new future evolutions of the code.

We see that the choice of one of these approach is meaningful, and should be decided early in projects. When people want to use libre software, they need to take care of the real price to pay to get things fitting their needs and benefit in the right way. In a case where it is a matter of integrating existing components to build new applications, not only would one have to decide what to get, or at what price (or what will be saved doing so), like in any traditional project. Another question, is whether one wants to fully play the "free software game", by giving back the contributions to the upstream projects used, as much as one receives from them.

In our case, we concluded that contributing to external free software projects was not only a way to play the game in a fair way, but also the only viable way to guaranty that the projects we relied on could benefit from our contribution in the future, for the good of all, and *first of all for us*.

Contributing may be as easy as sending constructive remarks about the product, giving patches for enhancements or bug fixes. But sometimes it requires a deeper involvement, which costs more, but which helps to adapt to unexpected evolutions of the external projects.

A.2 Hazards in libre projects

In the ProGet project, phpGroupware [6] was the key external source of trouble with respect to the management of our project. Unfortunately, just after the beginning of our project, when we had already chosen that technology, a *fork* of the phpGroupware project was launched by some of the main developers, who started the eGroupWare [13] project.

So all new developments on the phpGroupware codebase were momentarily postponed, and some time was needed to resolve conflicts, officialise the fork, and have

Proceedings of the First International Conference on Open Source Systems
Genova, 11th-15th July 2005
Marco Scotto and Giancarlo Succi (Eds.), pp. 113-118

the project going on again. The code-base was not impacted massively during the reorganisation (at least for the modules that we are depending on), but the procedures and the management of the phpGroupware project were clarified (this includes issues like ownership of contributions[6], quality control, etc.). The developers who decided to continue to work in the phpGroupware project were those clearly aware of these rules, and they agreed on the will to enforce them in the future.

For "outsiders" like us, who were willing to contribute, and arrived in such a time, it was quite difficult at the first glance to choose between continuing with the "historical" project, or switching to the new project. We spent lots of efforts, in addition to the development of our code, in talking with the developers, to try and make-up our minds on this issue.

Such hazards are common in free software communities, and a natural counterpart of basing developments on the latest "unstable" versions of products developed by free software teams which may be loosely structured and mainly composed of volunteer programmers. Even in a *well known* free software project, the leading developers can "lose" the support of parts of their community of users or contributors when such a fork occurs. Fortunately in our case, after some month of reorganisation, the project was back on its wheels with more clearly defined tracks, among which its adherence to the GNU project [14] standards.

We were bothered by the uncertainty on the future of phpGroupware, and by consequences, on our project. But, we were anyway very motivated in contributing our developments to the community, for instance to improve the LDAP authentication system in phpGroupware, or contributing to a reworked DAV VFS layer. And this motivation, strengthened by the choice of our contribution policy, has helped us wait until the project had clarified its procedures.

Since phpGroupware was meaning to clearly stay within the GNU project, we had a strong confidence that contributing to this one project was very important to ensure the durability of our contributions, even if it was longer and more difficult than expected initially.

A.3 New contributors integration process

In the case of phpGroupware, the quality is essential and contributing without respecting some common rules is not much appreciated. That's the case in most free software projects, even if they may seem unorganised from an external perspective. In the case of phpGroupware, a new developer first enters the project by submitting "tiny patches" that can be easily integrated when they respect the coding standard. After some time, the mainstream developers know the newbie better, and they grant him/her the right to directly commit into the main code-base by giving him/her access

---

[6] parts of the dispute was caused by issues relating to ownership of some contributions, in which respect the GNU project has clear established rules.

to the CVS[7] repository. At this stage the developer needs to be very careful with the way he/she commits things, and respects the policy for new code. For example, one should ask first to the "Dev" mailing-list advices about some pieces of code, or ask for testing on other parts. This process is helped by tools like the `savannah.gnu.org` [15] platform, and the "Trackers" it provides.

B. Myths about free software projects' quality

We wanted to build our project on top of libre software, so we needed to be careful to the quality of these projects. By quality we mean :

packaging: is the software easy to install on standard distribution?

extensibility: how easy is it to extend the project, add new components, ...?

openness: are the maintainers open to contributions? At what conditions?

maintainability: how are patches integrated and how easy is it to upgrade the software?

re-usability: is the framework easy to use? Is there a well defined API? Are the components reusable? Is the code easy to read?

documentation: Is there any documentation for user, developer, webmaster? Are there tutorials?

Even when you have a good idea of what the software claims to be able to do, you need to take care of the its real quality. Some times it is the truth, and that is great. But only when you look in the heart of the code, can you distinguish parts of the program that are *well maintained*, along with others that are here due to a *one-day contributor*. These pieces of code were done for the one day contributor's own usage, and are not used by the rest of the project. This code may be working at the date of the contribution, but unfortunately, as the project is evolving, this kind of code, in most cases, becomes obsolete. If there are plans to use this part in the in-house project, this kind of piece of software may threaten its viability.

As explained above, some of the core elements we would rely on were already chosen, because we already had some experience with them. That was the case with phpGroupware. For Twiki, we knew that researchers in GET were already acquainted with it. And Sympa is also used by the GET to manage lists for students and teachers.

But it was only after a while, when we "digged" into the code of phpGroupware that we were able to assess its real quality. It was not always good, some of its components suffering this "one-day contributor" syndrom. It is thus not only necessary to contribute, but contributing in a lasting way may help ensure that the code's quality is the same in all contributed parts.

C. Our contributions towards a more generic and stable phpGroupware system

For the interested readers, we describe here our contribution to the Simple Object Access Protocol

Proceedings of the First International Conference on Open Source Systems
Genova, 11th-15th July 2005
Marco Scotto and Giancarlo Succi (Eds.), pp. 113-118

(SOAP) Engine in the API. To make the integration of Sympa and phpGroupware easier to maintain, we could use SOAP and Web Services Description Language (WSDL) to communicate. The SOAP part was old and not able to work with Sympa. So we decided to rework this part of the API (thus, contributing to enhance phpGroupware API) to get things working. The contributed code should be in the next release of phpGroupware, allowing us to "forget" it in our custom patches.

The same is true with the VFS layer in the phpGroupware API. In prevision of a possible migration to Apache 2 a test was driven against the new DAV module. Some problems appeared and were solved by our work. We took time to improve the whole VFS layer, using the object-oriented approach to factorise the code between the different back-ends. By contributing this extra code, we get a benefit (no more patches to maintain for this part of the code) and help the project (more people are able to use the DAV back-end with fewer problems).

Giving our own code that was developped for our in-house application to a project like phpGroupware is in fact a good deal. This relieves us from the need to build specific patches that could be broken when the project releases new versions. And if our code is successful and is used in the upstream project, this gives us precious feedbacks (bug reports, patches) and helps us in doing things better. Also, helping the main project gives it more chances to be used by more people, and thus gives a relative security about its chances to survive in the software competition.

Because we were aware of these facts, we did convince the research office and administration of GET[7] to let us do so that our contributions could be integrated in the phpGroupware project. According to the GNU project policy on contributions, that requires that copyright is transfered by the programmer to the FSF [16], so that the FSF can defend the project in case of a trial.

D. Difficulties

It is clear that the effort done on phpGroupware to move parts of our code to the upstream project was high, and that such involvement was not possible for all the other projects we used. In the case of Sympa we just reported some minor bugs (and no heavy modifications were necessary). In the case of Twiki we did some bug reporting about the Koala skin and the modifications done were easily included due to its specificity.

In these cases it is clear that these parts will be harder to maintain, even if the modifications are less intrusive than in the phpGroupware case. But we needed to do a choice. Our resources were limited and being active on these others project at the same level as on phpGroupware could have lead to a counter productive situation.

---

[7] although the GET is a public funded structure, it is not so accustomed to "give away" its Intellectual Property rights.

VI. Conclusion and perspectives

This paper presents the lessons learned from the *ProGet* project, in which we integrated existing libre software programs to build a new in-house groupware platform, and adopted a policy for contribution to these external libre projects.

It claims that even if the initial cost for building an in-house application from existing libre software is greater when the project members contribute to the external libre software project, this is what should be done for a better mid-term maintenance of the in-house application.

It also tries to demystify free software reputation in terms of quality. The ProGet project, helped us understand libre software aspects and the significant role of the social processes of the software developers.

We were able to contribute efficiently to the key application used by our in-house project, in this case phpGroupware. We could have been more efficient in terms of collaboration, with the other components. We hope that we can continue developments on this in-house project, and reuse the phpGroupware modules we contributed to develop, in order to create a new release of PicoLibre. This would provide a new libre generic collaboration platform for others to use. By applying our policy for contribution, we hope that PicoLibre will become much more maintainable, and could remain usable with future new versions of its core components.

REFERENCES

[1] The GNU general public license (GPL).
    http://www.gnu.org/copyleft/gpl.html.
[2] What is wiki.
    http://wiki.org/wiki.cgi?WhatIsWiki.
[3] E. Cousin, G. Ouvradou, P. Pucci, and S. Tardieu. Picolibre: a free collaborative platform to improve students' skills in software engineering. In *IEEE International Conference on Systems, Man and Cybernetics*, Hammamet, Tunisia, October 2002. IEEE.
[4] The picolibre project.
    http://www.picolibre.org/.
[5] The sourceforge hosting platform.
    http://sourceforge.net/.
[6] The phpgroupware project.
    http://www.phpgroupware.org/.
[7] Concurrent versions system (CVS).
    https://www.cvshome.org/.
[8] OpenLDAP : An open source implementation of the lightweight directory access protocol.
    http://www.openldap.org/.
[9] Apache hyper text transfer protocol (HTTP) server.
    http://httpd.apache.org/.
[10] Web based distributed authoring and versionning (DAV).
    http://webdav.org/.
[11] The sympa mailing list manager.
    http://listes.cru.fr/sympa/.
[12] Twiki : an enterprise collaboration platform.
    http://twiki.org/.
[13] The egroupware project.
    http://www.egroupware.org/.
[14] The GNU project. http://www.gnu.org/.
[15] The GNU developement repository.
    http://savannah.gnu.org.
[16] The free software foundation (FSF).
    http://www.fsf.org/.

Proceedings of the First International Conference on Open Source Systems
Genova, 11th-15th July 2005
Marco Scotto and Giancarlo Succi (Eds.), pp. 113-118