Prerequisites For Enterprises To Get Involved In Open Source Software Development

Tilman Seifert[1] and Thomas Wieland[2]

[1] Institute for Computer Science
Technical University Munich
Garching, Germany
seifert@in.tum.de
[2] University of Applied Sciences Coburg
Coburg, Germany
thomas.wieland@fh-coburg.de

Abstract. Open source software development follows a development process that is defined by various rules and conventions as well as the set of tools that is being used, depending on the particular project. Commercial organizations which plan to enter the open source scene usually use quite different processes and tools. They have to integrate the tools used by the community into their development environment and adjust their processes accordingly. This is no trivial task. In addition, there are important cultural differences between Open Source and commercial projects. This paper describes the prerequisites that are necessary for a company to join "the OSS community" and become an accepted player there. Our argumentation is backed by a survey about tools and processes which we conducted among a couple of larger open source projects.

1 Introduction

More and more enterprises become interested in open source software (OSS). First they simply use it, e.g. to achieve cost reductions. But soon their developers discover that some OSS library or application could be an excellent basis to build their own product upon. So the involvement in the OSS scene gets deeper and more complicated. Finally they might even start thinking about releasing parts of their software as open source.

One of the first problems companies encounter during this engagement is that the tools and processes are often quite different from what they normally use. For getting a smooth curve to the OSS world instead of risking a crash on the straight line, some organizational prerequisites are thus necessary. But which tools are relevant? Which adjustments of the processes are necessary? These questions have certainly no general answers. They depend on the actual project and may vary considerably. To get a broader overview what is really used we conducted a small survey among a couple of larger OSS projects. So before giving some hints about the recommended prerequisites we'll have a look at the results of this survey.

2 Survey on Tools and Processes

In this survey we sent a questionnaire about tools and processes to members of 15 larger OSS projects. Seven of them replied with detailed answers: ACE/TAO, Debian, Enhydra, MySQL, KDE, Python, and Zope. Often the replies came from project leaders or key persons, like David Axmark, board member of MySQL AB, or Matthias Kalle Dalheimer, "elder statesman" from KDE. So we could cover an average project size of 294 developers from 31 countries, writing 1.7 million lines of code per project in average.

2.1 Questions on Processes and Decision Making

Formal processes are quite rare in OSS development. Only MySQL which stems directly from a commercial development uses an elaborate process (Scrum-like, see e.g. [Martin et al., 2001]). Others take elements from light-weight processes like extreme programming [Beck, 1999] or agile development [Cockburn, 2001] and enrich them with their own decision mechanisms. For example, KDE gives the contributors the maximum freedom by allowing "he who writes the code gets to decide".

Phases like in common iterative development cycles [Boehm, 1988] are also less strict. Most projects simply consider each release an iteration. MySQL manages iterative development by running four parallel code trees. In Zope, on the other hand, phases of an iteration are more clearly identified in form of feature cycles: proposal, voting, API agreement, tests, code.

Tasks in software development processes are carried out in an organized and repeated way. Unsurprisingly, there are not many tasks of this type in OSS projects. Most emphasis is laid on version and release management, for which most project have developed rules and procedures. But also quality assurance and test management are organized tasks in a majority of projects. More or less unknown, on the other hand, is risk management.

In OSS development hierarchical structures with a direct and strict assignment of tasks are widely unknown. Tasks are either immediately picked by any volunteer who is willing to fulfill it or are assigned in mutual agreement. They usually arise from the suggestions of project members or in common discussions.

Another element of open source software projects are key persons who usually founded the project and are now reigning as "benevolent dictator for live". Examples are Guido van Rossum in Python and Jim Fulton in Zope. They have a kind of popal edict rights and veto rights for all major decisions in the project. This model is not so unfamiliar for small and medium enterprises where there is usually a single boss that holds similar power. Given the historic background of MySQL, it is no wonder that the CTO/CEO of MySQL AB has similar rights, too.

2.2 Questions on Tools

When asked about tools, the field was not as diverse as one may expect. A small number of standard tools seem to evolve in the open source development scene. The tools that were most mentioned are listed in Tab. 1. It is remarkable, however, that some types of

tools, which are very common in commercial development teams, are largely unknown in OSS projects – even though there are free programs available. A typical example for this observation are UML design tools.

Area of usage	Tools used
Requirements	Text editor, Wiki
Use case collection	Text editor, Wiki (if at all)
Design	Umbrello, mailing list
Config management	Automake, autoconf, ZConfig
Version management	CVS
Testing	JUnit, Bugzilla
Packaging	Zip, ant, tar, RPM
CRM	Bugzilla, Usenet, mailing list
Bug tracing	Bugzilla, PHPBugtracker
Project management	Web, mailing list

Table 1. Tools used for specific tasks

2.3 Conclusion of the Survey

In addition to the problems mentioned above, some more threats were mentioned in individual answers of the survey. One was that the projects heavily depend on volunteer committers. If they are lacking, the project gets stuck or cannot at least reach the goals envisioned by the team.

More people, on the other hand, increase the amount of communication. This fact represents a major challenge for OSS projects with strong growth, which have to find a way to integrate more people constructively into the team. This problem is also very common in commercial development (see e.g. [Brooks, 1995]), but has different aspects there. Software projects are scalable especially in debugging and testing. More testers usually mean less errors. When testing is done in a parallel and redundant way, complete exhaustiveness can almost be reached. Some commercial projects with a stringent QA process may also run on this level of quality (with the respective resources necessary) – in many projects, however, exhaustive testing is confined due to budget or time pressure.

The reasonable organization of the testing is the next challenge. When the code base and the mutual dependencies increase, the number of necessary tests explodes. The fact that “enough” testers are at hand is not sufficient. The testing tasks have to be split and assigned (or picked) in order to ensure that all tests are really carried out.

The survey as a whole shows that OSS projects have many problems in common with classical, i.e. commercial, software development. But it is interesting to see how they solve them in a different way.

3 Similarities and Differences of OSS and Commercial Development

Many companies are already involved in some sort of OSS project and have adopted the cultural principles lived in the OSS community. Even more companies, however, have become interested in open source software development and demand to know what they have to do to participate. The simple answer: "Go along and contribute" will certainly not lead to a success because the goals and the expectations of the community and the company usually differ considerably.

In the following we will outline some organizational challenges that companies are faced with when getting in contact with open source. These challenges are about different processes and different tools as well as about more subtle cultural differences which a company needs to be aware of before getting involved in any OSS project.

3.1 Release Planning Process

The release planning process is important both for commercial and OSS projects. However, commercial projects typically feel time-to-delivery pressure and are driven by their next delivery deadline, while OSS projects can follow their own ideals about quality or features.

An effective software development process is characterized by a couple of properties that can be found in lightweight process models like XP [Beck, 1999] or in more heavyweight models like CMM-conform processes [Paulk et al., 1995]:

- Planned activities: Every step and every decision is planned (a priori) and traceable (a posteriori).
- Transparency of the process and the current state of the project for every stake holder.
- Planned communication: Information flows are defined.

These properties are realized by defining a role concept in order to assign tasks and responsibilities, and by defining a document concept in order to achieve the desired transparency and to avoid friction due to incomplete information. Note that this is common to commercial projects as well as to OSS projects, for OSS projects need some strict organizational policies to take advantage of the potentially high number of code contributors (cf. sec. 2.1).

Still, since incentives and the environment differ between commercial and OSS projects, these properties will look quite different. Commercial projects need more planning, like resource planning, negotiations with the customer etc. and therefore have more management overhead.

For OSS projects on the other hand, the major goal is to keep the organizational overhead to a minimum, because the volunteers prefer to contribute code instead of management activities. Most OSS projects solve this by introducing a strict release management process, as our survey has shown, too. This process is transparent to anyone and allows for contribution from a large number of developers (volunteers as well as paid developers). This way it is easy to install review processes that guarantee high code

quality, and at the same time the individual working habits of the developers don't have to be changed – a great advantage in a highly distributed development environment. The downside is that development stays fairly code-centered and advanced modeling tools and techniques can not be used. (See sec. 3.4 on the issue of tools.)

A company planning to join an OSS project needs to be aware of these differences in project organization. See also [Deng et al., 2003] for an analysis of the OSS release management process and its implications for OSS development in a commercial environment.

3.2 Quality Assurance

The quality of the produced software is of course an issue for both OSS and commercial projects. Most OSS projects apply strict reviews on code contributions before they are used in a branch of the code base. As opposed to many commercial projects, developers often do not have write access to the repository, and so the code has to pass a review by someone who can actually check in the code. This way, the QA process runs continuously and parallel to the development process.

This habit turns the disadvantage of the highly distributed development environment and the more difficult communication via mailing lists etc. into an advantage that pays off in form of high code quality.

3.3 Leadership by Competence

Open source projects depend on volunteers and have generally little chance to select their members. But for the acceptance of suggestions and the right to make check-ins to the main project tree it is often required that the developer in question has proven competence by his previous contributions. Many projects regard it as an important achievement that only excellent members can take over key functions of the project ([Fielding, 1999], [Mockus et al., 2000]).

Although this should be done the same way in companies, the picture in reality is often different. Project leadership is in numerous companies a hierarchical level, where people are promoted by other than technical criteria. It may happen that project leaders are lacking technical competence and have little reputation even among their own team members.

Successful OSS projects do not simply benefit from contributions of arbitrary developers as Raymond describes it (see, for example, [Raymond, 1999]). As Gauthier points out in [Gauthier, 2000], these projects are supported by extremely talented developers who are among the top 5%. The more popular and well-known a project is, the more developers are attracted.

A company that wants more than just sending in some sporadic contributions should acknowledge this principle. Especially if some influence on the directions of the project is envisaged, only the best programmers of this company should be assigned on this task. Useless or erroneous contributions will soon be rejected and turn out to be counterproductive for the goals of the company.

3.4 Different Tools and Standards

Large OSS projects often comprise several hundred developers, spread on many subprojects. On all parts of the graphical user desktop KDE, for example, there are currently 500 to 700 programmers from more than 50 countries involved. Together they have written more than three million lines of code. To produce such an enormous amount of software is only possible if there is a consensus about standardized tools as well as project and code structures. For the central implementation tasks the versioning system *CVS*, the configuration management *make* and the GNU compiler *GCC* are the tools of choice in many OSS projects. These tools are open source software themselves, thus everybody can easily get and use them (see table 1).

Hardly any OSS project takes advantage of modern modeling tools and techniques. Reasons are on one hand of practical nature; the wish for a standardized tool chain helps to exchange code and ideas very easily. On the other hand, it depends on the project culture. We often find the attitude "code rulez" – OSS developers tend to express their ideas directly in terms of code.

In commercial development, each project may opt for another technology leading to another set of tools. So the chances for code re-use from one project to another are usually very limited, even within one company. For participating in an open source community, it is hence essential to introduce the tools used in this community in the company. As this step can mean a thorough change of tools and habits, its necessity and benefits should be made clear to the developers in the beginning. Special training should be scheduled and some time should be allowed to become familiar with the new development environment. So approaching an OSS community can be seen by a commercial development department just like any other migration task. To achieve a rather smooth transition, a step by step procedure should be followed:

1. Explain the organizational and technical benefits of the new OSS tool set.
2. Train the developers on the new tool set.
3. Conduct small less critical projects with the new tool set.
4. Port the central application to the new environment.
5. Integrate the OSS and contribute enhancements and experiences.

If the approach to the community is too steep, many conflicts and misunderstandings will be the consequence of the harsh impact (see Fig. 1). On the other hand, both sides can benefit from a smooth approach leading to a common future development. It heavily depends, however, on the priority and intensity the enterprise is willing to invest in the open source software. Note that the situation is certainly different when the company cannot join an existing community, but has to start a new one from the beginning.

Besides the tools, the way of writing code (so-called "coding conventions") play an important role, too. Other programmers will only understand all code written for a particular project easily, if it has been written with the same comment rules, same naming schemes for identifiers, indentations, etc. A similar look of all code is not only a desirable goal, but should be enforced as much as possible. Ample code reviews, for example, one of the big advantages of the OSS methodology, are much more effective if the reviewer can concentrate on the algorithms and is not distracted by a peculiar style.

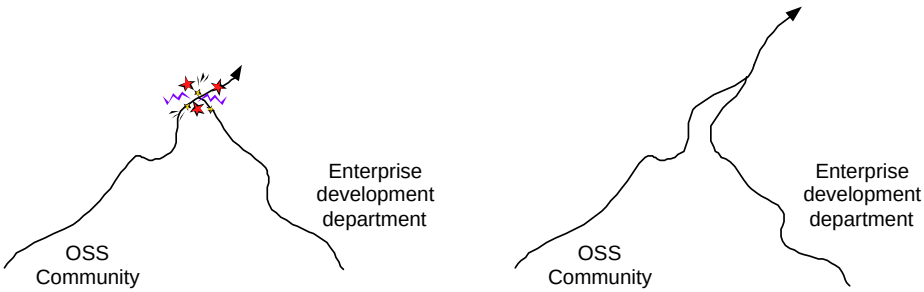

Fig. 1. Harsh and smooth approach of enterprise development to an OSS community

Moreover, involvement in open source means abandoning commercial libraries. If some code depends on a proprietary library, all prospective developers are forced to get a license of this library even to compile the code. This is completely inacceptable for open source software and a taboo in the entire scene. If the company plans to build a commercial product on top of the OSS which inevitably depends on a closed-source library, the access must be clearly encapsulated. The open source software must under all circumstances be compiled and run without the library. Even the access layer should be separated in the architecture such that it does not affect the open source code.

3.5 Motivation of Developers

It is obvious and well-known that the motivation of OSS developers is quite different from the one of commercial programmers (see, for example, [Hars and Ou, 2000], [Hars and Ou, 2001], [Hertel et al., 2003], [Markus et al., 2000]). A company that begins to cooperate with an OSS project must, however, be aware that the motivation paradigm of the community will also influence the motivation of the employed developers. They usually have more fun as they now feel that they are not only working for themselves, but are part of a larger community. And they feel prouder of their work because its quality is evaluated and appreciated by other programmers [Gauthier, 2000].

As the FLOSS study revealed [Wichmann, 2002], the motivation for developers to work on open source software is often the desire to learn and establish new skills. This observation holds true for voluntary free-lancing programmers as well as for employed developers, both participating in an OSS project. For the latter group, the typical task assignment by hierarchical superiors or the usual salary incentives therefore seem to be less efficient than relaxed time tables without tight schedules and the freedom to pursue their own ideas. Our own survey described in sec. 2 showed similar results.

If the encounter with an OSS community should have a win-win outcome, the respective company ought to change its development and communication culture. While the motivation of the developers shifts a little, approaching the “typical community programmer”, more or less naturally and inevitably, the work conditions also have to be adapted in order to avoid nipping this self-motivation in the bud. The so-called “geek” culture [Pavlicek, 2000] dominating most OSS projects requires much more openness, unbiased and complete information sharing, as well as mutual trust. Since such projects

are steered mostly by technically qualified individuals, typical management processes usually fail. A cultural change, at least in the team directly involved in the project as well as in its immediate environment, is thus essential. In this sense the involvement in Open Source does not only affect the programmers, but also the mid-level management. Unfortunately, the latter group can usually be convinced much harder from the benefits and the necessity for some organizational changes as described above.

The availability of an appropriate communication infrastructure with free and unlimited access to e-mail, usenet, web sites, and CVS repositories is on the other hand a matter of course.

4 Recommendations

It is virtually impossible to give universal recommendations, for all OSS projects are unique and differ from each other considerably. So the primary advice to a company that is interested in getting involved in a particular project is: "Learn as much as possible about the culture and tools of this project!" The notion of culture here includes not solely the communication culture, but also the decision procedures, the key persons, and areas of potential conflicts.

Summarizing the discussion in the previous section, the following recommendations may be given additionally:

- Assign some highly knowledgeable programmers to the project.
- Adopt the tools and standards of the community, also for internal sub- and related projects.
- Abandon commercial libraries.
- Give the developers the freedom to establish and contribute their own ideas, especially by relaxing the schedule and the management structures.

The self-motivation of the developers is one of the most crucial factors here that is also the hardest to achieve. It largely depends on the freedom with respect to time schedules, side-line developments, and realization of unusual ideas. Changes on a technical level can normally be implemented much easier than the ones on an organizational and structural level. Half of the way is already reached when the management acknowledges that there is no other choice.

5 Conclusion

For many companies, engagement in OSS projects seems to be appealing for a number of reasons. But the decision to join an OSS project requires some serious commitment by a company. Usually processes have to be changed or adapted, and a new set of tools must be introduced which requires some training. In the case that a company considers to release parts of their own software as Open Source, maybe some commercial, third-party libraries have to be encapsulated or even exchanged, which might lead to expensive architectural changes.

These efforts can pay off for both the company and the Open Source community. Reuse and public reviews of the software often lead to high code quality. Higher quality of the base technology might even increase the productivity of the company. This might leverage the profitability of the company, or it might shift the business focus from supplying base technology development to the development of higher level functionality and services.

Acknowledgments

This work was supported by the German Federal Ministry of Education and Research (BMBF) in the project "NOW – Utilization of Open Source Software in Business and Industry". The authors thank their partners at Siemens Corporate Technology, Siemens Business Services, and 4Soft for fruitful discussions and hints.

Moreover, the authors would like to express their thanks to all participants of the survey, who rapidly gave detailed answers, leading to many interesting insights.

References

[Beck, 1999] Beck, K. (1999). *Extreme Programming Explained: Embracing Change*. Addison-Wesley, Reading, MA.

[Boehm, 1988] Boehm, B. (1988). A spiral model of software development and maintenance. *IEEE Computer*, 21(5):61–72.

[Brooks, 1995] Brooks, F. (1995). *The Mythical Man-Month*. Addison-Wesley, Reading, MA.

[Cockburn, 2001] Cockburn, A. (2001). *Agile Software Development*. Addison-Wesley, Reading, MA.

[Deng et al., 2003] Deng, J., Seifert, T., and Vogel, S. (2003). Towards a Product Model of Open Source Software in a Commercial Environment. In *3rd International Workshop on Open Source Software Engineering*. ICSE 03.

[Fielding, 1999] Fielding, R. (1999). Shared leadership in the apache project. *Communications ACM*, 42(4):42–43.

[Gauthier, 2000] Gauthier, C.-A. (2000). Open source software quality. In *CASCON 2000*.

[Hars and Ou, 2000] Hars, A. and Ou, S. (2000). Why is open source viable? a study of intrinsic motivation, personal needs and future returns. In Chung, M., editor, *Proceedings of the 2000 Americas Conference on Information Systems*, pages 486–490.

[Hars and Ou, 2001] Hars, A. and Ou, S. (2001). Working for free? - motivations for participating in open source projects. In Sprague, R., editor, *Proceedings 34th HICSS Conference*.

[Hertel et al., 2003] Hertel, G., Niedner, S., and Herrmann, S. (2003). Motivation of software developers in open source projects: An internet-based survey of contributors to the linux kernel. *Research Policy*, (Special Issue on OSS).

[Markus et al., 2000] Markus, M., Manville, B., and Agres, C. (2000). What makes a virtual organization work? *Sloan Management Review*, (Fall):13–26.

[Martin et al., 2001] Martin, R., Schwaber, K., and Beedle, M. (2001). *Agile Software Development with Scrum*. Prentice-Hall.

[Mockus et al., 2000] Mockus, A., Fielding, R., and Herbsleb, J. (2000). A case study of open source software development: The apache server. In *Proceedings of the 22nd International Conference on Software Engineering (ICSE 2000)*, pages 262–273. ACM Press.

[Paulk et al., 1995] Paulk, M. C., Weber, C. V., Curtis, B., and Chrissis, M. B. (1995). *The Capability Maturity Model: Guidelines for Improving the Software Process*. Addison-Wesley.

[Pavlicek, 2000] Pavlicek, R. (2000). *Embracing Insanity: Open Source Software Development*. SAMS Publishing.

[Raymond, 1999] Raymond, E. (1999). *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. O'Reilly, Sebastopol, CA.

[Wichmann, 2002] Wichmann, T. (2002). Free/libre open source software (floss): Survey and study. Technical report, Berlecon Research, Berlin.