Software Configuration Management Issues with Industrial Opensourcing

Lars Bendix
Department of Computer Science
Lund University, Sweden
bendix@cs.lth.se

Tero Kojo
Nokia
Helsinki, Finland
tero.kojo@nokia.com

Jan Magnusson
Sony Ericsson Mobile Communications AB
Lund, Sweden
jan.magnusson@sonyericsson.com

Abstract—**The industrial involvement in Open Source Software projects is increasing. More and more companies are turning their proprietary code into open source, contribute actively to the development of Open Source Software projects or/and use Open Source Software products as (part of) their own products. Software Configuration Management provides the infrastructure that is the foundation for any type of software project. It facilitates the co-ordination and communication between the various participants on a software development team. Many problems and challenges from industrial involvement in Open Source Software projects have been identified in experience reports and research papers. A good part of these can be related to either absence of Software Configuration Management or a mismatch between what is done and what is needed for a particular setup. Many companies are used to Software Configuration Management in a homogeneous and localized setup and are confused about how to behave when the setup changes to a heterogeneous and distributed setting. In this short paper, we investigate Software Configuration Management lessons learned from the industrial participation in Open Source Software projects of two major telecommunications companies. We address what challenges can appear and discuss strategies to deal with these challenges.**

Open Source Software; Software Configuration Management; industrial experience

I. INTRODUCTION

Open Source Software (OSS) has always had a lot of attention from industry. However, in recent years the character of this attention has changed. In the beginning OSS products were looked at as competitors to proprietary products – or as products that you could use free of charge. Lately more and more companies have started to look at OSS as a software development method – a method that can also be adopted and used by commercial companies.

The origin and history of open source is open for debate [17]. However, it seems to start in the early eighties from Richard Stallman's Free Software movement whose aim is to promote the universal freedom to create, distribute and modify computer software [16]. In the nineties the term OSS comes into use and is represented by key projects like Apache and Linux. Key OSS projects from the start of this millennium like Eclipse and Firefox[1] have their origin in code produced and/or sponsored by commercial companies – and one of today's flagship OSS projects, Android, seems to be a primarily commercial participation project. The plethora of more or less permissive "free to modify and distribute" OSS licenses will *not* be dealt with in this report. We will leave these legal aspects to others and focus on the fact that the source code is open – and therefore possible to modify.

Today the direct and indirect involvement of industry in OSS projects is quite high. More and more companies are turning their proprietary code into open source [18]. About one third of the 300 most active projects on SourceForge[2] had industrial involvement in 2007 (and one third of those projects were founded by a company) [5]. And more than 75% of the contributions to the Linux kernel come from (people who get paid by) companies [6]. OSS has moved from a community of individual developers towards a community of commercial organizations as industry has adopted the OSS development method in a way that has been coined "opensourcing" [18].

Software Configuration Management (SCM) is one of the fundamental capabilities that should be in place in any software development project and a pre-requisite for being able to carry out better all activities on a software project [12]. It is traditionally considered to consist of four activities (configuration identification, configuration control, configuration status accounting, and configuration audit) [7]. It is looked at as a management tool that can help in guiding a project, maintain the integrity of the product and keep the quality under control [11]. However, there is a different, but very related, perspective where it is seen as a developer-centred discipline that focuses on how to maximize the productivity of

[1] Firefox is an OSS project, which delivers a "professional" product that even naïve end-users can download, install and automatically update.

[2] As of May 2011 SourceForge hosts close to 300.000 OSS projects.

a team of developers by providing support for the co-ordination and communication within the team [3]. Exactly what kind of SCM is needed on a project and in particular how it should be implemented depends very much on the specific context.

A good part of the problems and challenges that have been identified with industrial involvement with OSS development [14], [15] can be related to issues with SCM. In a previous work, Asklund et al. have investigated SCM and OSS [2]. Their work was an analysis of how different SCM activities were carried out by the OSS community at the turn of the millennium and what lessons industry could learn from that to apply on their own projects. Many things have happened in the past decade and in this paper we re-visit SCM and OSS. This time with special emphasis on what has changed with the involvement of industry in OSS projects. We analyze experience from the involvement of two major telecommunications companies in open source projects from both usage and participant sides. Between them the two companies cover a wide spectrum of "participation modes". Furthermore, we supplement this by including experience reported in experience reports and research papers.

Now that they have embraced OSS as a development method, industry needs to know how SCM should be performed in this development method. In the following, we first categorize different ways of industrial involvement is OSS projects, then we report on experience and strategies for the different categories and discuss advantages and drawbacks of different strategies before we finally draw our conclusions about the lessons learned.

II. Industrial Participation in OSS Projects

Industrial involvement in OSS projects can come in many forms and ways – from doing a little testing to managing a whole project, from using a single component as-is to customizing a whole project. In this short paper, we do not want to discuss ontologies or taxonomies, but we need a framework to structure and organize the presentation and discussion of our experience and strategies.

In this section, we will briefly sketch our categorization of different forms of industrial involvement in OSS. The categories fall in two main dimensions: contributions to OSS projects and usage of OSS code/products. Bonaccorsi et al. list (without much further explanation) three levels of involvement in OSS projects: collaboration to code development; provision of code or protocols; and project co-ordination [5]. We will use their categorization more or less unchanged for our contribution dimension. There are other aspects, like the size and complexity of the project or the velocity and size of changes to the code, which might also be used for categorization. However, we find that the sketched structure serves our purpose and will include other aspects where we find them relevant for our discussion of suitable strategies.

A. Service participation

In Bonaccorsi et al. this type of participation is "collaboration to code development in different phases and at different events, like bug fixing, testing or offering services" [5]. It is a very simplified type of participation that is characterized by being passive (like testing) or contributing only very limited amounts of code (like bug fixing). You do not have commit privileges to the code repository and submit code changes as patches.

Even if you contribute many bug fixes over a longer period of time we will consider your contributions as service participation. Our division is not so much by the quantity of contributions or the duration of the participation but by the nature of what and how you contribute. For this reason you will need and use the same tools and processes no matter how often and how long your participation lasts.

B. Development participation

"Provision of code or protocols; for example communication protocols used to share information among different devices" is how Bonaccorsi et al. characterize this type of participation [5]. We put emphasis on the fact that what is contributed is substantial amounts of new code that implements some new functionality. If you contribute a re-write of some existing functionality we would be inclined to consider that as "service participation". The reason is that there is already (as is also the case for bug fixes) a precise "specification" of what has to be done (the existing code) and test cases should already exist. When we are talking about new functionality, we are also talking about new "requirements" that have to be specified and all that follows (test cases etc). You are a much more active participant and in some cases you may have direct commit access to the source repository – in other cases you may still have to submit your contributions as patches even though it is new functionality.

There is no obvious, clear-cut distinction between the "service participation" and "development participation" categories. In some moments you might participate by contributing services while in other moments your contributions (and way of working) will be that of development.

C. Owner participation

This type of participation is characterized as "project co-ordination" by Bonaccorsi et al. [5]. In the standard OSS terminology that would be more like a moderator or a module owner – or in traditional software engineering terminology a project manager or a product owner. Your role is to see to that there is an agreed upon road map for the project. You will not have fixed, stable resources to manage as a traditional project manager, but will have to co-ordinate the resources that are offered to you at any given time by the volunteers.

Moderators can be organized in a hierarchy for big OSS projects. Sometimes a moderator role will be carried out by the same person for a long(er) period of time – other times moderators, especially at lower levels of the hierarchy, will change at irregular intervals. Sometimes a moderator will also contribute code or bug fixes – most of the time they will neither have the time nor the interest. Owner participation might be a misleading term since nobody actually owns the code since it can be easily cloned. We were considering "co-ordination participation" as a better term, but settled on "owner participation" since it puts more emphasis on the project than

on the code. In fact the owner of the project (the moderator or coordinator) is the one who decides what gets accepted into the project and what is rejected. If people do not agree, they are free to clone the project and start a new one where they become owners.

D. As-is usage

For this type of usage, the code of the used OSS modules is not changed in any way and is used as it is. In reality you do not even need the source to be open, as you could have used the binary, but it is convenient to be able to actually read the source code for "documentation". Sometimes you will have to write some "glue code" to use the modules as part of a larger product other times the modules interface without any "glue".

Using open source products off-the-shelf will not be covered since it is similar to buying a normal product – just "cheaper". To complete the "spectrum" of usage, there could also be the "no-use" type of usage. You do not have to actually "use" things from an OSS project to contribute – you may want to contribute to "push" certain issues. Since you will not have any "usage" (SCM) problems we leave out this category.

E. Modified usage

Here you exploit, on a smaller or larger scale, the possibility to go into the source code and change it. In effect a local variant of the project is created, which corresponds very much to what happens if the original OSS project is cloned and has to a large degree the same consequences. Sometimes you would like go off on your own "tangent" and leave the original OSS project behind – other times you would like to still have the original OSS project's additions, modifications and improvements to the code that you do not modify.

III. SCM Experience and Strategies

In this section we will present, analyse and discuss our experience from practical industrial involvement in several OSS projects. We will draw on experience from all contribution types as well as from all usage types.

In general you will have to familiarize yourself with the tools and processes that are used on a particular OSS project. It will be the owner of the project who, together with the active community on the project, decides what tools to use and how the processes should be.

A. Service participation

One of the services that you may render an OSS project is testing. If there is no easy way to download and install the binary for the product, you will have to get the source code and build the binary yourself. The way "outsiders" carry out testing in an OSS project varies a lot. Some people will perform a very careful test of the product including both black-box and white-box (since the source code is open) testing techniques – others will just use the product, experience a problem and report that. In either case, there should be a bug reporting system where you can report the results of your testing. The fact that the source code is open gives "outsiders" the possibility to give more information on the possible cause of the reported bug. Ideally, it should also be possible to contribute test cases for everyone else to use (similar to the unit- and acceptance tests from agile projects).

Another service you can perform is code review, which is often not done as an explicit activity on "traditional" OSS projects, but is more explicitly used on "industrial" OSS projects. More often than not there is no tool for doing the actual code review. However, there should be a structure in place that will tell you what modules and versions are in need of a review and make it possible for you to contribute the result of your review.

However, the most common service that you can offer is the contribution of bug fixes. There are two slightly different reasons for contributing bug fixes: fixing a problem *you* have experienced, and fixing a problem *the project* has experienced. In the first case you experience a problem when using the OSS product and since the source is open you might be able to fix the problem. It may be convenient to know if someone else is working on fixing the same problem, but it is not essential. In the second case, there should be a bug tracking system in place. In that system you can see what bugs are prioritized by the project owner, and you will have the complete information available that allows you to work on and fix the bug.

Since you usually will not have direct commit access to the repository, you should familiarize yourself with how to create a patch for the change that you have made and how to send it to the right moderator.

Asklund et al. found that the moderators on an OSS project very easily become a bottleneck [2]. This means that if you want to get your bug fix into the code, it is not sufficient that your fix is of good quality – equally as important is that it is easy for the moderator to apply, review and test your patch. One common problem reported was that very often patches are dropped if they cause merge problems – you can avoid that by staying as close as possible to the latest version of the project.

B. Development participation

Contribution of new functionality differs from contribution of bug fixes in the way that the latter are quite easily accepted (if they are of good quality) even if they are not discussed and prioritized beforehand. Most often new functionality will have to be discussed before it is even worth trying to contribute. Asklund et al. found that an important reason for moderators to reject contributions with new functionality was that it did not take the project in the right direction [2]. So you will have to familiarize yourself with the OSS project's communication infrastructure, use that for proposing your idea for some new functionality – and take part in the discussions it will create. Sometimes you will find new functionality that has been proposed – and "accepted" as wanted – but no-one has come around to implementing it yet. Such a "requirement" would be easy to get through if the implementation is of good quality.

When you have completed the implementation of the new functionality you are ready to submit. If you do not have direct commit access to the source repository, your situation is quite similar to the one in "service participation". However, if you have commit privileges, you will have to merge your changes

with the latest version of the code. In either case your contribution will have to be integrated with the rest of the code.

Since there will be many people working in parallel on the same project, there will be the "double maintenance problem" identified by Babich [3]. In effect the work of each single developer will form an independent line of work. Experience shows that the longer these lines exist, the more they will grow apart – and the more they grow apart, the more difficult they will be to merge because of the possibility of conflicting changes. A simple solution to that problem would be to adopt the strategy of "continuous integration" [10] to avoid as much as possible difficult merge situations. This strategy was also identified by Asklund et al. as the one used and encouraged by the OSS projects they analyzed [2]. In Deshpande et al. [8] it was hypothesized that OSS projects had picked up the "continuous integration" strategy from Extreme Programming [4] and the agile movement. They investigated differences in commit size and frequency pre- and post-1998 (when the Extreme Programming idea first came out) and found no significant differences. Their conclusion was that the OSS community had not picked up "continuous integration" as a way of working. We find that very improbable and tend towards the alternative, which they discard with no further motivation, that the OSS community was already using the "continuous integration" strategy way before it was popularized by the agile community. We find support for that claim by the fact that the "copy-merge" working model together with the "long transaction" model [9] in CVS, which is an OSS product used (in particular previously) to support OSS projects, actually penalizes you if you do not use an integration strategy similar to "continuous integration".

When you are contributing some new functionality, you may be tempted to "throw in" a couple of bug fixes as well – in particular if you discover some bugs during the implementation of the functionality. It is discouraged to mix functionality and bug fixes in the same contribution as it is also discouraged to put more than one bug fix in a single contribution. It will increase the probability that your contribution is rejected. First of all because the moderator's task will become more difficult if he is not able to keep things logically and physically separate. Second, because a problem in one part of the contribution will cause the whole contribution to be rejected. Finally, if you put more things into your contribution it will take longer to produce and you will become more prone to the double maintenance problem.

C. *Owner participation*

The owner of the OSS project will have to set up the tooling and processes used on the project. In general you should make the "entrance fee" as cheap as possible for the other participants. Unlike in a company where people might work for years, people who participate in OSS projects work for much shorter periods of time – sometimes even as little as a one-off contribution. When that is the case, it is not possible to justify a lot of work from people in trying to understand how things work and are organized on this particular project. That tends to lead towards a preference for simple and easy to understand tools and processes and towards tools that people might already be familiar with and processes that they might already be used to from other OSS projects.

The owner should also provide people who can follow the incoming contributions and make sure that they are prioritized (in case of bugs and new requests) or properly integrated (in case of bug fixes or new functionality). These people will be the moderators and in the case of a company starting an OSS project the first moderators might come from inside the company, but in the long run moderators will be selected from the community based on shown merits. A moderator will receive contributions from people and integrate them into the project if the contribution is deemed good. In effect the moderator will act both as the chairman of the Change Control Board of a traditional project and as the technical integrator of the code [2]. So it is easy for a moderator to become the bottleneck and everything should be done to make him work as efficiently as possible. It should be easy for him to apply the patches and to run a quick "smoke test" to see if they work.

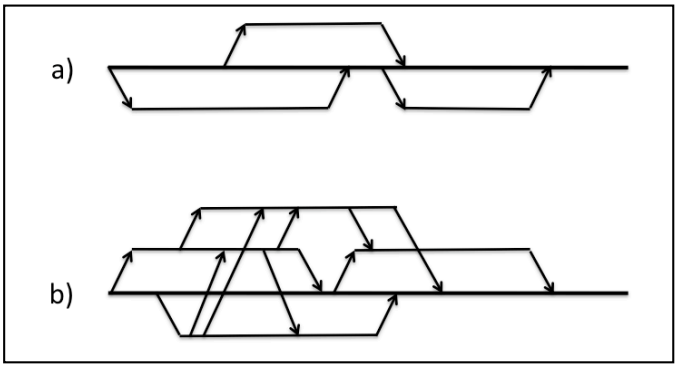

Figure 1. Ways of working.

There are many different ways the parallel work of contributors can be organized. In figure 1a, it is shown how a low velocity project could be organized. Each contributor branches off a separate line of development that is later on merged back to the mainline – either by the developer of by the moderator. The advantage of this model is its simplicity and it will be supported by just about any version control tool. However, as the volume and velocity of changes grow the way that people tend to work does not follow this simple pattern. They will often need to collaborate temporarily with other people as shown in figure 1b. One might want to get some code (bug fix or functionality) from someone else before it has been integrated in the mainline. This way of working looks more complex and will create merge problems for many version control tools, but tools with powerful merge tracking will be able to handle this situation.

A company that owns an OSS project must know how to act as an owner. If the company is afraid of "letting go of control" it will run the risk of scaring away contributors [13]. Real care should be taken to "respect" the spirit of the OSS community in the day-to-day management, including handling the requirements process and the long-term goals of the project.

D. *As-is usage*

Even if you use the code "as-is" there will be new versions of the code that you use. You will have to decide on a strategy

for whether you want to update to the new versions or not – and if you want to update, then how often. If you decide to update, you will have to test/review to see whether your "glue code" and the rest of your code still works as expected with the new versions of the OSS component that you use.

E. Modified usage

In the case that modifications are minor localization, tools like CVS are perfectly capable of handling that by the use of the "import" command. In effect the OSS project is imported on a vendor branch and the localization changes are placed on the main trunk, as shown in figure 2. When a new version of the OSS project is imported, it will be possible to merge it to the mainline and in effect have the localizations applied to the new versions. Sometimes you will experience merge conflicts if the localization touches code that has also been modified in the new version of the OSS project.

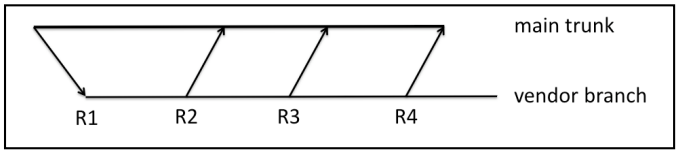

Figure 2. Handling "private" adaptations.

In a more realistic usage scenario you are not only making localization changes, but also changes that for some reason are not contributed back to the OSS project (they may have been rejected or not accepted yet). In this scenario the simplicity of figure 2 will break down. Depending on the extent and the number of changes it might be handled by more sophisticated branching strategies [1]. However, a better strategy is to use a version control tool that supports the change set model [9] (like git). In that case your localization – and other – changes become logical changes that can be (re-)applied to a given version of the OSS project to create your "personalized" product. This is particularly useful when we are dealing with code changes (eg. bug fixes) that will eventually make it into the OSS project and therefore would have to be removed from the main trunk in figure 2 – an operation that is very difficult in most version control tools, but very simple in a tool that supports the change set model.

IV. CONCLUDING REMARKS

Now that industry has embraced the OSS development method by participating in and/or running projects the "open source" style they also need to adopt and understand the SCM strategies that go with the OSS development method.

When setting up an OSS project processes and tools that are familiar from other OSS projects are important so potential contributers have a low "entrance fee". Equally important is to realize that the moderator is a key person. He works as the chairman of the Change Control Board and does the integration of the contributions. He must have excellent technical skills and design skills and big projects might need several moderators organized in a hierarchy to avoid bottlenecks.

Participants in OSS projects should know how to create and send patches to the moderator(s). Contributions should be kept small and logically separate to make it easier for the moderator to check and integrate contributions.

When you use an OSS project, you should try to contribute your changes back to the project. This will give you goodwill in the community and will make it easier for you to maintain your own product as it will deviate less from the OSS project.

REFERENCES

[1] B. Appleton, S. P. Berczuk, R. Cabrera, and R. Orenstein, "Streamed Lines – Branching Patterns for Parallel Software Development" in proceedings of the 1998 Pattern Languages of Programming Conference, Monticello, Illinois, August 11-14, 1998.

[2] U. Asklund and L. Bendix, "A Study of Software Configuration Management in Open Source Software Projects", IEE Proceedings – Software, Vol. 149, No. 1, February 2002.

[3] W. A. Babich, Software Configuration Management: Coordination for Team Productivity, Addison-Wesley, 1986.

[4] K. Beck, "Extreme Programming Explained: Embrace Change", Addison-Wesley, 1999.

[5] A. Bonaccorsi, D. Lorenzi, M. Merito, and C. Rossi, "Business Firms' Engagement in Community Projects – Empirical Evidence and Further Development of the Research", in proceedings of the First International Workshop on Emerging Trends in FLOSS Research and Development, Minneapolis, Minnesota, May 21, 2007.

[6] J. Corbet, G. Kroah-Hartman, and A. McPherson, "Linux Kernel Development – How Fast it is Going, Who is Doing It, What They are Doing, and Who is Sponsoring It", Linux Foundation White Paper, December 2010.

[7] M. A. Daniels, "Principles of Configuration Management", Advanced Applications Consultants Inc., 1985.

[8] A. Deshpande and D. Riehle, "Continuous Integration in Open Source Software Devekopment", in proceedings of the International Conference on Open Source Systems, Milan, Italy, September 7-10, 2008.

[9] P. H. Feiler, "Configuration Management Models in Commercial Environments", Technical report, CMU/SEI-91-TR-7, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, March 1991.

[10] M. Fowler and M. Foemmel, "Continuous Integration", http://martinfowler.com/articles/originalContinuousIntegration.html, September, 2000, accessed July 20, 2011.

[11] ISO 10007:2003, "Quality management systems – Guidelines for configuration management", 2003.

[12] M. C. Paulk, C. V. Weber, B. Curtis, and M. B. Crissis, "Capability Maturity Model for Software, Version 1.1", Technical report, CMU/SEI-93-TR-24, February 1993.

[13] M. Shaikh and T. Cornford, "'Letting go of Control' to Embrace Open Source: Implications for Company and Community", in proceedings of the 43rd Hawaii International Conference on System Sciences, Kaui, Hawaii, January 5-8, 2010.

[14] K.-J. Stol and M. A. Babar, "Challenges in Using Open Source Software in Product Development: A Review of the Literature", in proceedings of the Third International Workshop on Emerging Trends in FLOSS Research and Development, Cape Town, South Africa, May 8, 2010.

[15] K. Ven and H. Mannaert, "Challenges and strategies in the use of Open Source Software in Independent Software Vendors", Information and Software Technology, Vol. 50, Issue 9-10, August 2008.

[16] Wikipedia, "Free Software Foundation", http://en.wikipedia.org/wiki/Free_Software_Foundation, un-dated, accessed July 20, 2011.

[17] Wikipedia, "Open Source Software", http://en.wikipedia.org/wiki/Open-source_software, un-dated, accessed July 20, 2011.

[18] P. Ågerfalk and B. Fitzgerald, "Outsourcing to an Unknown Workforce: Exploring Opensourcing as a Global Sourcing Strategy", MIS Quarterly, Vol. 32, No. 2, June 2008.