Configuration Management for Open Source Software

Ulf Asklund
Department of Computer Science
Lund Institute of Technology
Box 118, S-22100 Lund
Sweden
Email: Ulf.Asklund@cs.lth.se

Lars Bendix
Department of Computer Science
Aalborg University
Fredrik Bajers Vej 7E, D-9220 Aalborg Øst
Denmark
Email: bendix@cs.auc.dk

Introduction

Open Source Software (OSS) projects have a seemingly anarchistic way of organising projects and a set-up (many, distributed developers) that is usually considered difficult to handle within the field of configuration management. Still they manage to produce software that is of at least as high a quality as that produced by Conventional Software Development (CSD) projects.

We have investigated more closely what they actually do, and why they are so successful. The goal of the study was to describe their underlying configuration management process, thereby making it explicit, so it can be followed in case others (like commercial companies) want to start an OSS project or a project having similar characteristics. We also analysed to what extent their success is due to a good process, good tools or simply to outstanding people participating in OSS projects. Based on this, lessons could be learned from OSS and possible transferred to conventional ways of developing software. We interviewed key people from three OSS projects (KDE, Mozilla and Linux) to obtain data for our study.

We need a framework to compare the way CSD projects and OSS projects carry out the configuration management task. Traditionally configuration management is divided into: configuration identification, configuration control, configuration status accounting and configuration audit. This approach is primarily a top-down view at what configuration management is and is better suited for a manager perspective. Open Source development, however, is not about management but about development, and we need a framework that can analyse configuration management from that perspective. A more developer-oriented view at configuration management is: version control, build management, configuration selection, workspace management, concurrency control, change management and release management. In this position paper, we do not present our full study, but just the change management task, which is where OSS configuration management differs mostly from CSD configuration management.

Change management processes compared

The reasons for changes are multiple and complex and handles all changes in a system, both perfective, corrective, and adaptive changes. Change management includes tools and processes that support the organization and track the changes from the origin of the change to the approval of the actually implemented source code. Figure 1 and 2 depict the process of change management in CSD and OSS respectively.

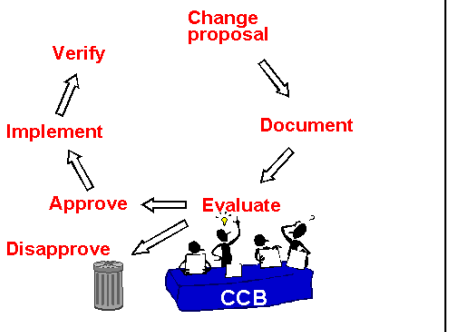

Figure 1. Conventional change process

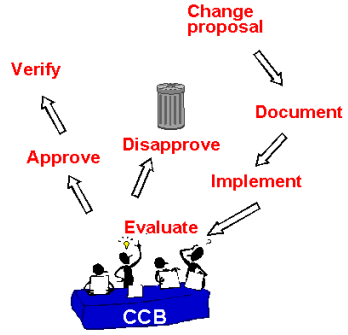

Figure 2. Change process for OSS

In CSD, a change should only be carried out as the result of an approved change request. When a change is initiated, change requests are created to track the change until it is resolved and closed. The change control board (CCB) analyses the change request and decides which action is to be taken. If the change is approved, the change request is filed to the developer responsible for implementing the change. When the developer has performed the change its status becomes "implemented" and a test is performed. When the subsequent new release is to be built, the change control board decides which changes are to be included and the customer receives a patch including documentation of all the changes made.Various tools are used to collect data during the process of tracking a change request. It is important to keep traceability between the change request and the actual implementation - in both directions. Change management data can also be used to provide valuable metrics about the progress of project execution. From this data it can be seen which changes have been introduced between two releases (a set of change requests). It is also possible to check the response time between the initiation of the change request and its implementation and acceptance.

In OSS, the evaluation of change proposals is not explicit, if it is there at all. Anyone can propose a change and most often changes are not even proposed before an implementation is submitted directly. Change proposals might be prioritised implicitly or explicitly, but an OSS project cannot assign tasks to developers - everyone works on what he chooses. Two slightly different processes exist depending on whether contributions have to be sent to a moderator or if you can apply your changes directly to the repository through your write access. In both cases, however, it is the same overall process that is followed. An idea for a change is conceived, it is implemented and tested, it is submitted as a patch or applied directly on the repository, and finally the implementation (and sometimes the change idea itself) is evaluated through testing, review and discussion. The final evaluation may result in the patch being rejected by a moderator or a change to the repository being reverted by a co-ordinator. Usually write access to the repository is given only to trusted developers, so cases where a change to the repository is reverted are rare.

Linux, which is the prime example of an OSS project with moderators, gets patches submitted which are then worked through by the moderator. Patches are reviewed in multiple steps before testing, and only then inserted in the repository. Contributions that are found ill-designed or have not so sound ideas are rejected at reading time. If the idea is good but the code is bad, the contribution usually undergoes a few iterations of review before testing. If a contribution is rejected, there is sometimes feedback to the author.

Some projects, like Mozilla, have a mixed approach of module owners and direct write access to developers. The modules owner has the right to reject patches. In most cases, any of the developers with write access to the repository can make changes in most places of the code. There is no fixed policy; rather each module owner sets the contribution and change policy for his module. Sometimes, the module owners can pose a bottleneck to the processing of contributions. In projects that work exclusively through co-ordination only, it seems like most changes are accepted immediately, and very few, if any, are rejected. The few patches that are received are handled by the co-ordinators.

Most change management problems seem to be caught at the review stage. Contributions are often tested via code reviewing and special run time tests. However, formal testing is not always used. Sometimes when a change has been made, developers simply use the new code. Developers that have submitted many good patches are more trusted, and their contributions make their way into the repository quicker. Accepted contributions show up immediately in the repository.

Even though wish lists and lists of bugs are kept, bugs and change proposals seem to be fixed in a somewhat arbitrary fashion. Changes are kept track of using detailed lists, in order for willing users to test new features. Mail and newsgroups are used to communicate wish lists, bugs, and changes and to discuss the general development of the project.

Important factors in OSS success

Here we will analyse some of the configuration management factors in an OSS project. We have divided the analysis into three categories: tool support, process, and people. For each category, we discuss what we have found to be the most important properties in an OSS project. The aim is to make these important properties explicit, explained, and possible to copy for new OSS projects. Examples correlated to change management are:

- (Tools) All versions should, of course, be stored in the server. To provide awareness of what have happened to the project, it should be easy to browse through the history of the project. It is also possible that some bad

submissions sometimes reach the code base, which may lead to difficulties building the system. In these cases the tool should provide support to revert the entire change containing the bugs, i.e. not only some specific files that first have to be detected.

- (Process) The change management process must be appropriate for the task. An effective way to reduce the complexity of change management is to not maintain old releases. Instead all development, both bug fixes and new requirements, are committed directly to main. Otherwise several branches have to be maintained, which now can be avoided.

- (People) The largest single risk of an OSS project is that the moderator becomes a bottleneck. Such bottlenecks not only delay the awareness and usability of the application developed, but also break some of the advantages of the long transaction model of the used CVS tool. One important property of the model is that the developers always commit a tested (and working) configuration, if needed after several iterations of 'update-merge-test' within the private workspace before a successful commit. The developer can, however, only update and test from the common repository and can not access the submissions not yet processed by the moderator. If these contain modifications not consistent with the new submission the moderator has to send back the submission and the developer has to update and re-send it again.

Transfer of lessons learned

Some of the factors highlighted in the previous sections can be used also within CSD projects, at least after some adjustments, others can not. How configuration management is handled within OSS is, however, no 'silver bullet' solving all kinds of problems and there may (even) be some lessons that OSS has to learn. In this section we will focus on the transfer of knowledge and best practices, mostly from OSS to CSD but also vice versa. Examples correlated to change management are:

- It is important to *protect your code base*. Traditional CSD puts a lot of effort to classify and give priority to change requests and to decide whether they should be implemented or not, but does less to protect the code base from bad implementations. A role similar to a *moderator or co-ordinator* in OSS might be a good idea.

- Important to the success of OSS is self assigned tasks and the fact that the developers almost always also are users and testers of the system. This is unfortunately hard to copy to CSD development, but if possible a project should strive towards a similar process. If possible they should use the developed application. If this is not possible extensive testing should be performed, e.g. in the style of XP.

- Most OSS projects lack the control and visibility of the 'requirements' and change requests implemented and the ones still on the wish list. In CSD this is often managed by separate tools and considered one of the most important activities of configuration management. Most OSS projects had probably benefited from an updated wish list and a *better traceability between a change request (wish) and the actual change* made to the code.

- In CSD it is also important to set the correct priority on all requirements, depending on severity, how much it costs to implement, importance to different markets, etc. This is harder to do in OSS since all tasks are self assigned, i.e. each developer makes their own priorities independent of how other users/developers set their priority.

Conclusions

In this position paper, we have looked at change management. Other configuration management tasks also studied (but not presented here) are version control, configuration selection, workspace management, and concurrency control.

Interesting future work is to attempt an evaluation of how efficient OSS development really is. Additional topics to study specifically related to the configuration management tasks are: how long does it take to respond to a bug report, how much double/useless work is there in submissions, how many contributions are lost because of bad change management, can the OSS model (for configuration management) also be used for the start-up phase of a project?