Adoption of Configuration Management in the Industry: Strategies and Lessons Learned

Cláudia Werner, Chessman Corrêa, Rodrigo Santos, Marcelo Schots

System Engineering and Computer Science, COPPE, Federal University of Rio de Janeiro
Rio de Janeiro, Brazil, 21.945-970
{werner, chessman, rps, schots}@cos.ufrj.br

and

Leonardo Murta

Fluminense Federal University
Niterói, Brazil
leomurta@ic.uff.br

and

Flávio Lyra

Brazilian Electrical Energy Research Center, CEPEL
Rio de Janeiro, Brazil
flavio@cepel.br

Abstract

Maintenance represents an important activity in software industry as it is the one that takes the biggest effort among Software Engineering activities apart from its high cost. In this sense, software configuration management helps to overcome some difficulties related to software maintenance such as the lack of product knowledge and negligence in maintenance activities. This paper presents a configuration management deployment process based on standardization of practices and tools, taking into account the software development organizational culture. The results of its application in an industrial case are discussed.

Keywords: Configuration Management, Software Maintenance, Software Process.

1. INTRODUCTION

Maintenance plays an important role in the life cycle of software. In the early 90s, many organizations allocated at least 50% of their financial resources to the maintenance activity [20] and, in the present decade, this percentage has reached 70% and become one of the challenges in Software Engineering (SE) [2]. According to Polo *et al.* [19], the reason for such high cost is partly due to the nature of this activity, marked by unpredictability. This reveals the importance of research into the process of maintenance and its application in industry. However, Paduelli & Sanches [17] say that there is a lack of works aimed at documenting the problem areas in the software maintenance field.

Kajko-Mattsson *et al.* [9] justify the difficulties of this activity via the following points: (1) problems in error diagnosis, related to investigative processes to detect the real cause of a problem prior to correcting it; (2) absence of documentation; (3) lack of product knowledge; (4) negligence in conducting software changes (i.e., lack of management); and (5) low esteem on the part of the maintenance agents as a result of the previous points. Aiming at contributing to remedy (or reduce) these points, Configuration Management (CM), in the context of SE, is the discipline that allows the evolution of different artifacts produced along software development process in a controlled and organized fashion [6].

CM appeared in the 50s to meet a need of the US aerospace industry: controlling the changes to documentation as related to the production of war planes and spacecrafts [11]. In the two decades after that, its scope was enlarged, and it also included software artifacts [3]. However, the focus still remained on military and aerospace applications.

Only with the rise of standards, such as the first IEEE Std 828 [8] version in the early 80s, CM had its focus enlarged to other domains.

The use of CM systems during software development allows an adequate control over artifacts generated and modified by different stakeholders. CM is not aimed at defining when and how work product modifications should be carried out. This issue is part of development process. However, CM activities take place as an auxiliary control and monitoring process [21].

From a management point of view, CM is divided in the following functions [8]: (i) configuration identification, (ii) configuration control, (iii) accounting of the configuration status, (iv) configuration assessment and revision, and (v) release and delivery management. On the other hand, from a development point of view, CM is divided into three main systems: (i) *modification control*: stores data on change requests, reporting them to interested and authorized stakeholders; (ii) *version control*: controls configuration items evolution in a disciplined, distributed and concurrent manner; and (iii) *construction management*: automatizes the system building process and the baseline release. It is important to point that these perspectives relate to each other in an overlapping manner, which means that the five functions described in the management perspective can be implemented by the three systems described in the development perspective, added with manual procedures if needed.

According to Murta [16], it is important to see that CM is part of the main initiatives towards an improvement in process and product quality, such as ISO 9000, CMMI, ISO/IEC 15504, and MPS.BR. Thus, several organizations that produce software have been continuously implementing CM practices in their productive chain. However, despite the relevance of CM support to maintenance, less than 25% of Brazilian companies of a 2,500 universe used it in 2005 [13].

Since CM is a key factor to software development and maintenance, this paper aims at describing a process to implement CM based on the standardization of practices and tools, taking into consideration the organizational software development culture. The process was applied at Brazilian Electrical Energy Research Center (CEPEL) along the last three years. Additionally, this paper consists in an extended version of a paper previously published in the proceedings of the VII Workshop on Modern Software Maintenance, co-located to IX Brazilian Symposium on Software Quality [25].

Besides this introductory section, the paper is organized in five other sections: Section 2 describes the process proposed to implement CM; Section 3 discusses the case study produced at CEPEL, covering a brief characterization of the projects and some of the difficulties found; Section 4 presents some statistical data extracted from the repository regarding the project development; Section 5 shows an experimental study (survey) executed with the stakeholders in order to obtain additional, qualitative data about the proposed process for supporting conclusions; finally, Section 6 provides the lessons learned and some final considerations.

2. PROPOSED PROCESS FOR CM IMPLEMENTATION

The use of well-defined plans and processes aids in the standardization of activities and generated artifacts in software development organizations. Moreover, the control over developers' work and over the data produced is increased through the systematization of the steps needed for each activity execution, helping towards predictable and repeatable results. In this sense, a process is proposed for CM implementation, as shown in Figure 1.

Figure 1. Process proposed for CM implementation

The first activity in the process is **to prepare the environment**. This activity holds two tasks: choosing the CM technologies and their deployment in the organization. The choice can be determined by the characteristics of the projects, size of the teams, and resources offered by the CM systems.

The second activity in the process is **to create awareness in the interested parties**. In this activity, the stakeholders are duly informed of the changes that will be carried out. The reasons and benefits to be obtained are clarified. Other

goals of this activity are to keep all involved parties motivated and avoid (or minimize) the resistance to change prior to the execution of the next process activities. It is worth pointing that this activity should entail the members of all the projects that will adopt CM.

The third process activity is **to diagnose the project**. The first task of this activity is to hold a meeting with the consultants and the project team. The goal is to allow the project team to present their project to the consultants, allowing them to understand the project structure and its development history. Moreover, the meeting allows the consultants to identify the characteristics that are relevant to the application of CM in the context of the project, such as: (i) distributed and concurrent development, (ii) number of developers, (iii) project size, and (iv) previous CM practices. To avoid the overlooking of any important aspect of the project, a standard script is used as a reference. Following the meeting, the consultancy team prepares the diagnosis document that represents their project understanding. This document also includes an initial suggestion for the versioning repository structure, as well as for the implementation strategy to be used. Following the preparation, a new meeting is scheduled to present this diagnosis so that the project team can evaluate it and suggest adjustments.

The activities **training project team**, **planning CM** and **preparing for migration** can be carried out simultaneously. In the **training** activity, the project team is coached to assimilate the best CM practices, and to become familiarized with the tools to be used. At the same time, in the **planning activity**, the CM plan for the project is created. This plan serves as a reference guide for the CM-related actions that are to be applied along the project lifecycle. This plan also facilitates the integration of a new member in the project team, given that the policies adopted for CM are set. To facilitate the creation of the document, a template is used, based on the IEEE Std 828 standard [8]. This template can be adapted according to the particularities of the project. At the end of the activity, the plan is presented to those involved in the project in an evaluation meeting. In the **prepare for migration** activity, tests are conducted, as related to the migration of the project to new CM systems, considering a new organization structure of the repository, if needed. The tests serve as reference to identify issues related to the migration and for the preparation of a strategy for its execution.

After the project team has been trained, the CM control plan has been approved, and the migration strategy has been completed, the **migration** activity is executed. In this activity, the available artifacts are migrated to the respective CM systems chosen for the organization. Finally, with the completion of these stages, the **mentoring** activity is carried out where the consultancy team clarifies any possible issues of the project team as regards the use of the procedures in their real scenario. This stage is fundamental to ensure the effectiveness of the implementation of the CM mechanisms and to enable the project team to use them with safety and autonomy.

3. CM IMPLEMENTATION AT CEPEL

This section presents and discusses the process use in the implementation of version control at CEPEL along the last three years. CEPEL is an institution that is part of the Eletrobras System and has over 30 years in research and development on power generation, transmission and distribution. Most of the professionals at work in the research area of this institution have an Electrical Engineering background.

CEPEL has several software projects under development. Part of the projects was not under version control. Most of the time, this control was done by the developer, who normally organized prior versions under directories in file systems, doing backups in media such as CDs. The developers that already used a version control system (VCS) did not do so in a uniform way. Some projects used the Visual Source Safe (VSS) tool [14], while others used the Concurrent Version System (CVS) [7], and one of the projects used Subversion [4]. However, some issues were identified and reported by several users (especially in relation to the VSS tool, confirmed in a Microsoft article [15]), such as corrupted bases and a difficulty to manage access control. These problems led to the creation of a partnered project between CEPEL and Federal University of Rio of Janeiro aimed at implementing CM at CEPEL.

The CM implementation process described in Section 2 was applied to 16 projects in CEPEL, with two being large, seven being mid-sized, and seven being small. In this work, a small-size project is understood as having less than 100 files and one or two developers; a large-size project has over 10,000 files and a team with more than five developers; and a mid-sized project lies in between the two other project types mentioned. As it can be seen in Table 1, the two large projects used a VCS. Amongst the seven mid-sized projects, five used a VCS (71%). On the other hand, none of the seven small projects used a VCS. These data indicate that large and mid-sized project developers had more concern with the version control. It can also be seen that VSS was the main VCS adopted.

Table 1. Project characteristics

Size	None	SVN	CVS	VSS
Small	7	0	0	0
Medium	2	1*	2*	3
Large	0	0	0	2

*One of the projects used SVN and CVS for different subprojects.

Considering that the initial focus was the implementation of a standard VCS, the process was adapted to cover only this activity. Due to that, the preparation of the environment involved the choosing and deployment of the VCS and the adoption of version control plans.

In the environment preparation activity, Subversion was chosen as a tool for version control. This decision was based on the product robustness function, on the large number of existing users, and of the range of available support tools. It is important to note that this choice occurred in early 2007 when the distributed VCS still did not present sufficient robustness and usability for their adoption in a corporate environment. After the choice and implementation of Subversion, the activity to create awareness was carried out for the members of all the projects. After that, the implementation of version control started, for each one of the 16 selected projects.

The diagnosis activity was executed in the same way for all the projects. In the training activity, the team for each project attended a theoretical training course on Subversion and a practical training course with the tools to be used: SVNManager [22], TortoiseSVN [5], and Subversive [18]. At the same time of the training course, the prepare migration activity was done. The tests made in this activity helped to detect and solve many issues related to the repositories based on other VCSs, especially VSS. Also, the CM planning activity was executed to create the version control plan.

The following activity was migration. The project artifacts were migrated to the Subversion repository in CEPTEL. In the case of projects that did not use a VCS, the stage of migration consisted of the inclusion of project artifacts in the repository created for it. In some cases, a desire was expressed to include prior versions stored in media in the shape of directories. For that, the inclusion of the initial version was done and at each new version to be included a comparison was made with the immediately previous version so to store only the differences between versions (as it usually happens in an automatic manner with projects that already are under version control). For each version that represented a release, a tag was created. In the case of projects that used CVS or VSS, this stage involved a migration process from the respective base to Subversion. Finally, the mentoring activity was performed, from which the issues of the members of a team were solved.

4. ANALYSIS AND RESULTS

In order to allow analyzing the use of SVN repositories and CM best practices, focusing on the scope of VCS, the consultancy team adopted a two-dimensional analysis, by (1) using a tool for collecting and generating repository statistics regarding the project development and (2) by applying a survey to obtain additional, qualitative data for supporting conclusions (Section 5). Considering (1), the deployed tool is StatSVN [23], an open-source software which retrieves information from a Subversion repository and generates a report with tables and charts based on these data. The CEPTEL partner authorized the consultancy team to analyze the generated report.

The first verified question was:

- Q1: Is the repository being used by developers?

For answering this question, we first needed to answer two other questions:

- Q1a: When did the most recent commit happen? (the most recent the commit is, the better)
- Q1b: In general, how long is the period (length of time) between commits? (the shorter, the better)

Before answering these questions, it was important to check the number of committers per project, as well as the migration date – in some cases, the number of committers does not reflect the number of developers. This information is shown on Table 2, where the items marked with an asterisk (*) include a SVN login that was created for the consultancy team to import the project data into the repository. Note that some developers may participate in more than one project. Due to preserve sensible data, the project names were omitted. Although the end date from report period varies for each project, all the reports were generated in July 2010 (as the time of writing). Projects 15 and 16 could not be verified because the tool used for collecting data did not generate the report for them. Project 15 is a very large project, and the script used for generating the report crashed during its execution. The repository created for Project 16 seemed to be deleted or moved.

Figure 2 presents the lines of code committed by date for some of the projects. It is important to mention that some projects already used a VCS before using SVN. In these cases, the commit history was imported by the consulting team. For instance, this is the case of Project 14, which used Microsoft VSS [14]. Also, in this project, it is possible to see that the lines of code (LOC) decreased significantly in 2010. This happened because the team decided to keep some subprojects (that were already discontinued) out of the migration process.

One limitation of the tool used for statistics is that LOC counts are inaccurate for deleted or moved files (that is, when a file is deleted or moved, StatSVN does not track exactly how many lines it had and by whom they were committed) [24]. Although Figure 2 is a good way of checking the growth of each repository (in terms of committed LOCs), it does not allow verifying the lines of code contributed by each developer. This is shown in Figure 3 for Projects 04, 05, 06 and 08.

The number of developers shown on each graph may include a SVN login that was used for the consultancy team for importing project data into the repository (as can be seen on Table 2). Projects 07, 09 and 11 only contain the

commit that imports the project data to the repository, so they were not included in this part of the analysis. After investigating this issue, the consultancy team realized that Projects 09 and 11 were not active, and the migration was done for backup purposes. Also, the teams of both projects mentioned that these projects may be reactivated at some point. Project 07 is undergoing restructuring.

Table 2. Number of committers per project (ordered by migration date)

Project	Number of committers	Migration date	Report period
Project 01	3	December 2007	2007-12-06 to 2008-10-09
Project 02	6*	January 2008	2008-01-15 to 2010-07-09
Project 03	5	March 2008	2006-03-10 to 2010-05-19
Project 04	10*	August 2008	2000-09-08 to 2010-07-08
Project 05	8	October 2008	2008-10-07 to 2010-07-08
Project 06	37*	October 2008	1997-08-21 to 2010-06-14
Project 07	1	December 2008	2008-12-15 to 2009-09-08
Project 08	2*	July 2009	2009-07-21 to 2010-07-08
Project 09	1*	August 2009	2009-08-18 to 2009-08-18
Project 10	2*	October 2009	2009-10-02 to 2010-02-03
Project 11	1*	November 2009	2009-11-11 to 2009-11-11
Project 12	2*	November 2009	2009-11-12 to 2010-06-30
Project 13	13	December 2009	2005-04-27 to 2010-07-08
Project 14	13	January 2010	1997-10-17 to 2010-06-14
Project 15	Could not be verified	Could not be verified	Could not be verified
Project 16	Could not be verified	Could not be verified	Could not be verified

Figure 2. Lines of code per project

There are some interesting observations from this point of view. For example, Project 06 contains a lot of committers, and a detailed analysis could show how much and how long each developer participated in the project. Also, in Project 04, the most active committers have a considerable difference (in terms of lines of code) when compared to the other ones.

Figure 3. Contributed lines of code per developer on each project

Due to the scale used on Y-axis of each graph, it is not possible to observe small commits (i.e., commits of less lines of code). So, commit activity for Projects 05 and 08 (for demonstration) is presented in Figure 4. The first row of each project represents the whole team activity, and each subsequent row is related to a developer. In this figure, it is possible to answer Q1b question, by checking the intervals between dates.

In Figure 2 and Figure 3, it is interesting to note that the repository created for Project 08 does not seem to be used by the project team. However, by analyzing Figure 4, it is possible to see that commits were made in the last months – this can happen because the scale used on each graph may result in a biased analysis, so additional information is required. Also, this figure helps to identify whether only releases are committed. Another important point in some cases is that long periods with no commits may represent a vacation time or a pause on the project development, as some developers participate in more than one project (as mentioned before). Table 3 summarizes the analysis (the period of time between commits is considered short, medium or long by being compared to the same data from the other projects).

Table 3. Data summarization

Project	Is the repository being used? (Q1)	Last commit date (Q1a)	Period between commits (Q1b)
Project 01	No	October 2008	Long
Project 02	Yes	July 2010**	Short
Project 03	Yes	May 2010	Short
Project 04	Yes	July 2010**	Short
Project 05	Yes	July 2010**	Short
Project 06	Yes	June 2010	Short
Project 07	No*	September 2009	Long
Project 08	Yes	July 2010**	Medium
Project 09	No*	August 2009	Long
Project 10	Yes	February 2010	Medium
Project 11	No*	November 2009	Long
Project 12	Yes	June 2010	Medium
Project 13	Yes	July 2010**	Short
Project 14	Yes	June 2010	Short
Project 15	Could not be verified	Could not be verified	Could not be verified
Project 16	Could not be verified	Could not be verified	Could not be verified

* This project is not active.

** As the time of writing.

Figure 4. Commit activity per project

Another interesting question is how some CM practices are being used on each project. Our focus is especially on branching and tagging. The question Q2 relates to it:

- Q2: Are the project members using branching and tagging mechanisms properly?

Similarly to Q1, in order to answer Q2, we first need to answer three other questions:

- Q2a: Are branches being created?
- Q2b: Are tags being created?

- Q2c: Is a naming pattern defined for each project (or adapted by developers) being used?

Some projects had already a naming pattern because of client requirements, but in most cases these patterns were only related to final releases. So, the consulting team created and adapted naming patterns in order to fulfill each project's needs. The answers to these questions are presented in Table 4. The criteria used for analysis was:

- When neither branches nor tags were created, it was not possible to verify the using of a naming pattern (therefore, we could not verify whether these mechanisms were being used properly);
- When either branches or tags were created, branching and tagging were not being used properly;
- When both branches and tags were created, branching and tagging were used properly (as long as a naming pattern was being used).

Table 4. Use of branching and tagging

Project	Branching and tagging are used properly? (Q2)	Branches are being created? (Q2a)	Tags are being created? (Q2b)	A naming pattern is being used? (Q2c)
Project 01	Could not be verified	No	No	Could not be verified
Project 02	Not quite	Yes	No	Yes
Project 03	Yes	Yes	Yes	Yes (adapted)
Project 04	Not quite	Yes	Yes	No
Project 05	Could not be verified	No	No	Could not be verified
Project 06	Yes	Yes	Yes	Yes
Project 07	Could not be verified*	Could not be verified	Could not be verified	Could not be verified
Project 08	Yes	Yes	Yes	Yes (adapted)
Project 09	Could not be verified*	Could not be verified	Could not be verified	Could not be verified
Project 10	Not quite	Yes	No	No
Project 11	Could not be verified*	Could not be verified	Could not be verified	Could not be verified
Project 12	Yes	Yes	Yes	Yes (adapted)
Project 13	Not quite	No	Yes	No
Project 14	Could not be verified	No	No	Could not be verified
Project 15	Could not be verified	Could not be verified	Could not be verified	Could not be verified
Project 16	Could not be verified	Could not be verified	Could not be verified	Could not be verified

* This project is not active.

Note that we did not count how many branches and tags are being created because each project has its own development cycle and timeline, so this metric cannot be used as a unit of comparison. Based on this analysis, it is possible to note that, although most of the developers are using the repository, some concepts are not being properly applied. Especially in some cases, branching and tagging mechanisms do not seem to be absorbed. This can be due to a lack on consultancy training, developers learning or both, although it is not possible to infer anything based on the statistics. This leaded us to elaborate a survey for collecting qualitative data about the consultancy as a whole, in order to obtain additional data that can help us to better understand the scenario and verify strengths, deficiencies and improvement opportunities.

5. A SURVEY WITH STAKEHOLDERS

Considering the proposed process in the context of the sixteen target projects, some elements related to the developers and project managers' perspectives should be evaluated in order to confront the collected data by tools against the feedback provided by stakeholders. For example, (i) how are the process and tools being used after the configuration management implementation process (if they were modified or not)? (ii) are the teams satisfied with the process and tools? (iii) what are the improvements and difficulties? etc. So, an experimental study (survey) was conducted with the stakeholders involved in the projects in order to collect perceptions, analyze points of view and provide a feeling about the proposed process.

Using the GQM (Goal/Question/Metric) approach [1], the goal was to *analyze* the proposed CM deployment process, more specifically focused on version control *for the purpose of evaluation with respect to* verify some feedback elements related to the followed version control plan, realized improvements and difficulties, and safety from the developers and project managers *point of view, in the context of* the sixteen target projects at CEPEL. The survey process was composed by the following steps, based on [12]: (1) problem statement; (2) selection of experts; (3) elicitation of opinions; (4) aggregation of opinions; and (5) decision making.

5.1 Survey Phases

The adopted approach follows four phases: (1) defining, evaluating and validating the survey; (2) selecting and contacting the experts of the sample; (3) sending the survey invitation by e-mail, and collecting data; and (4) analyzing the data in a semi-quantitative way (aggregation of subjects' opinion and discussion about comments and observations). Two documents were generated: the plan document and the questionnaire document [10]. The questionnaire was composed by two sections: the subject characterization (4 questions) and feedback (6 questions, some of them depending of the stakeholder profile and/or requiring respective justifications, and the seventh one is open for additional points).

For the sample, 72 stakeholders were selected by convenience (non-probabilistic sampling) from the sixteen projects, based on data extracted from the Subversion repository. From these stakeholders, the sample was reduced to 70 due to invalid or non available contacts. The analysis of the judgment provided by the subjects' responses showed that no significant biases were introduced in the study. In addition, all subjects were equally weighted during the data aggregation since no significant difference was observed in terms of their credibility or importance. From the sample, 20 stakeholders answered the questionnaire in July 2010. This rate corresponds to 28.5% of effective subjects: (i) 85% of the subjects are developers, and 15% are project managers; (ii) 10% work in large projects, 45% work in medium projects, and 45% work in small projects; and (iii) effective subjects are involved in 81.25% of the sixteen projects.

5.2 Survey Results

After the first three survey phases, the feedback elements were analyzed in a semi-quantitative way (forth phase). Some results of the survey are shown in Figure 5 and Figure 6, asking about the use of Subversion and version control plan after the process deployment, respectively. As noticed, most of the subjects are using the Subversion infrastructure and following the plan developed during the process implementation, although adapted. In this case (and also in the negative one), the main modifications and reports were related to: (i) creating an external repository to support third-part developers; (ii) including non-intermediary versions into the repository, that is, each commit corresponds to a release; (iii) using no branches during development; (iv) using Subversion in new projects (e.g., the stakeholder left the team of the considered project); and (v) using the VSS and treating the Subversion as a periodical repository, since some plug-ins failed in the context of the IDE (i.e., Microsoft Visual Studio). Moreover, considering the subjects who are using the SVN (90% as presented in Figure 5):

- 60% create and maintain branches in the project repository;
- 60% create and maintain tags in the project repository;
- 65% check logs in the project repository;
- 75% use commit and check out in the project repository;
- 10% use Subversion only for backup.

Figure 5. Use of SVN after the process deployment

Figure 6. Use of the version control plan

When subjects were asked to provide their feedback according to their profile, that is, if they used a version control process before or not, some opinions can be highlighted. In the first case, a contradiction between two justifications were noticed when one subject presents that Subversion is easier and more complete when compared to VSS, despite the initial difficulties in changing the infrastructure and the process. However, another subject wrote that his team is still using VSS because it is easier to use it with the adopted IDE (Microsoft Visual Studio) – this subject had problems with the IDE plug-in and compatibility with Windows 7, and identified some deficiencies related to classical weak points of the IDE, presented by 11. So, this team uses Subversion only for backup. In the second case, a subject who has never used version control recognized the greater stability of Subversion infrastructure as well as the implemented process, saying that “*who is using VSS has a double work*”. In this context, analyzing the final perception, from 75% of satisfied subjects, 70% are feeling more safety after the version control process with

Subversion, against 5% that affirmed that Subversion is good as a corporate repository – but its instabilities with automatic merge in Windows 7 is a harmful factor.

6. CONCLUSION AND LESSONS LEARNED

Considering that the maintenance activity is gaining exposure in software industry as it requires greater effort amongst SE activities, apart from its high financial costs, CM can contribute to overcome some of the hurdles related to this activity such as the lack of product knowledge and the negligence in conducting changes to the software. In this sense, this paper presented a process proposed to implement CM based on the standardization of the use of practices and tools and on the concern with the organizational culture found in software development. From the proposed process, we studied its applicability, especially regarding version control, in an organization that has several software projects (CEPEL).

From the activity aimed at creating awareness of the concepts and the goal of CM in software development, the stakeholders could understand the importance of the implementation of a VCS for their projects. The teams responsible for small-size projects that manually controlled the versions did not still consider the lack of a support tool as a problem as the size of the team and the number of releases were relatively small and there was no concurrent work. However, thanks to the training effort and to the monitoring of the process, it was possible to show that the manual control of versions can become quite time-consuming as the project grows.

The duration of the stages for preparation and migration varied according to the size of the history in store (e.g., in another VCS, in folders, or on CDs), the number of project versions, and the type of VCS used. The projects that used VSS had greater difficulty in migration, particularly due to problems such as corruption in the repository base. The mean execution duration for this process was of three months per project. However, this duration varied for more or less according to some factors amongst which the complexity associated to the restructuring of the repositories (of teams that already used another VCS), the availability of the project team to hold the meetings, and the migration stage.

The adopted VCS works by default with the optimistic policy, that is, prioritizing the concurrent work. Developers who used VSS displayed some resistance with this feature as it uses, by default, the pessimistic policy (based on locking actions). This fact was circumvented by the possibility of using the same policy in Subversion. Apart from that, the choice of the concurrence control policy in Subversion can be done for each artifact in an independent manner. Finally, the resistance regarding the use of the new tools, by part of some teams, was overcome by carrying out the training course and the mentoring activity.

It is also important to note that the inclusion of a training stage in the concepts and practices of the CM area many times dissolved resistances and doubts on the implementation. Therefore, the success of this kind of work depends both on the qualification and awareness of those involved, and on the planning of the new CM structure, apart from its correct implementation. For future work, there is a possibility of using the proposed process in other CEPEL projects. Also, we intend to use the process to deal with the other CM systems (e.g. issue tracking and build tools) and apply it in other organizations.

Acknowledgements

The authors thank all CEPEL professionals that took part in this project.

References

- [1] Basili, V., Caldiera, G., and Rombach, H.D. The Goal Question Metric Approach. *Encyclopedia of Software Engineering*, 1994.
- [2] Bhatt, P., Shroff, G., and Misra, A.K. Dynamics of Software Maintenance. *ACM SIGSOFT Software Engineering Notes*. Vol. 29, No. 5, (September, 2004), pp. 1-5.
- [3] Christensen, M.J. and Thayer, R.H. *The Project Manager's Guide to Software Engineering's Best Practices*. IEEE Computer Society Press and John Wiley & Sons, 2002.
- [4] Collabnet. *Subversion*. Available from: <<http://subversion.tigris.org/>>. Accessed on 18/07/2010.
- [5] Collabnet. *TortoiseSVN*. Available from: <<http://tortoisesvn.tigris.org/>>. Accessed on 18/07/2010.
- [6] Estublier, J. Software Configuration Management: A Roadmap. In: *Proceedings of the Conference on the Future of Software Engineering, 22nd International Conference on Software Engineering*, Limerick, Ireland, pp. 279-289, June, 2000.
- [7] Free Software Foundation. *Concurrent Version System*. Available from: <<http://savannah.nongnu.org/projects/cvs/>>. Accessed on 18/04/2010.

- [8] IEEE. *IEEE Std 828 – IEEE Standard for Software Configuration Management Plans*. Institute of Electrical and Electronics Engineers, 2005.
- [9] Kajko-Mattsson, M., Forssander, S., and Olsson, U. Corrective Maintenance Maturity Model (CM³): Maintainer's Education and Training. In: *Proceedings of the 23rd International Conference on Software Engineering*, Toronto, Canada, pp. 610-619, May, 2001.
- [10] Kitchenham, B., Budgen, D., Brereton, P., Turner, M., Charters, S., and Linkman, S. Large-scale Software Engineering Questions – Expert Opinion or Empirical Evidence?. *IET Software*. Vol. 1, No.5 (October, 2007), pp. 161-171.
- [11] Leon, A. *A Guide to Software Configuration Management*. Artech House Publishers, 2000.
- [12] Li, M., and Smidts, C. A Ranking of Software Engineering Measures Based on Expert Opinion. *IEEE Transactions on Software Engineering*. Vol. 29, No. 9 (September, 2003), pp. 811-824.
- [13] MCT. *Quality and Productivity in Brazilian Software Sector*. Science and Technology Ministry, Informatics Policy Department, 2006. In Portuguese.
- [14] Microsoft. *Visual Source Safe*. Available from: <[http://msdn.microsoft.com/pt-br/library/ms181038\(VS.80\).aspx](http://msdn.microsoft.com/pt-br/library/ms181038(VS.80).aspx)>. Accessed on 18/07/2010.
- [15] Microsoft. *How to detect and fix database corruption errors in Visual SourceSafe for Windows 6.0 and in SourceSafe*. Available from <<http://support.microsoft.com/?scid=kb%3Ben-us%3B133054&x=6&y=11>>. Accessed on 21/07/2010.
- [16] Murta, L.G.P. *Configuration Management Applied to Component Based Development*. PhD Thesis, System Engineering and Computer Science Department, COPPE/UFRJ, Rio de Janeiro, RJ, Brazil, 2006. In Portuguese.
- [17] Paduelli, M.M., and Sanches, R. Software Maintenance Problems: Characterization and Evolution. In: *Proceedings of III Workshop on Modern Software Maintenance, V Brazilian Symposium on Software Quality*, Vila Velha, Brazil, pp. 1-13, Junho, 2006. In Portuguese.
- [18] Polarion. *Subversive*. Available from: <<http://www.polarion.org/index.php?page=overview&project=subversive/>>. Accessed on 18/07/2010.
- [19] Polo, M., Piattini, M., Ruiz, F., and Calero, C. Roles in the Maintenance Process. *ACM SIGSOFT Software Engineering Notes*. Vol. 24, No. 4 (July, 1999), pp. 84-86.
- [20] Schach, S.R., and Tomer, A. A Maintenance-Oriented Approach to Software Construction. *Journal of Software Maintenance: Research and Practice*. Vol. 12, No. 1 (January-February, 2000), pp. 25-45.
- [21] Softex. *MPS.BR General Guide – MPS Model and Reference Model (MR-MPS)*. SOFTEX Society, May, 2009. In Portuguese and Spanish.
- [22] Sourceforge. *SVNManager*. Available from: <<http://svnmanager.sourceforge.net/>>. Accessed on 18/07/2010.
- [23] StatSVN. *StatSVN*. Available from: <<http://www.statsvn.org/>>. Accessed on 18/07/2010.
- [24] StatSVN. *StatSVN User Manual*. Available from: <<http://wiki.statsvn.org/Default.aspx?Page=User%20Manual>>. Accessed on 18/07/2010.
- [25] Werner, C., Murta, L., Corrêa, C., Santos, R., Prudêncio, J. G., Fernandes, P., Schots, M., Cepêda, R., and Lyra, F. A Process to Implement Configuration Management in Industry. In: *Proceedings of VII Workshop on Modern Software Maintenance, IX Brazilian Symposium on Software Quality*, Belém, Brazil, pp. 1-8, June, 2010. In Portuguese.