Software Configuration Management over a Global Software Development Environment: Lessons Learned from a Case Study

Leonardo Pilatti
Pontifícia Universidade Católica
do Rio Grande do Sul
+ 55 (51) 3320-3558
lpilatti@inf.pucrs.br

Jorge Luis Nicolas Audy
Pontifícia Universidade Católica
do Rio Grande do Sul
+ 55 (51) 3320-3558
audy@pucrs.br

Rafael Prikladnicki
Pontifícia Universidade Católica
do Rio Grande do Sul
+ 55 (51) 3320-3558
rafael@inf.pucrs.br

ABSTRACT
Software configuration management is an important support activity in the software development process. In global environments, the software configuration becomes critical due to the characteristics of the distributed development (physical distance, cultural differences, trust, communication and other factors). The objective of this paper is to analyze the software configuration management in a global software development environment, identifying the main challenges. The results are based on a case study carried on at a multinational organization that has offshore software development centers in Brazil, India and Russia, and was recently recognized in the CMM Model level 2 in the Brazilian unit. The results suggest the necessity to adapt and implement some activities in the software configuration management process addressing the main existing challenges. These activities were identified as lessons learned, collected at the end of each project. The problems and the solutions adopted are presented, aiming to relate these solutions to the organization distribution level, considering the project team, users and customers.

Categories and Subject Descriptors
D.2.9 [**Software Engineering**]: Management – *software configuration management, software process models.*

General Terms
Management.

Keywords
Global Software Development, Software Process Improvement, Software Configuration Management

1. INTRODUCTION
The crescent globalization in business environments has affected the software development market [1]. Aiming competitive advantages as low costs, high productivity and quality in systems development, several organizations decided to distribute their development process inside or outside their countries. India, Brazil and Ireland, as well as several other regions offer fiscal incentives and availability of resources in software development. However, there are several challenges in distributing the teams in a software development environment. Cultural differences, time zone and communication medias, per instance, shall be analyzed to avoid negative impacts in the organization.

In this context, software configuration management (SCM) is also influenced by the team distribution. The SCM process, even in co-located environments, is pointed as critical to software development [2]. When dealing with team dispersion, difficulties tend to increase. Some authors ([2], [3]) define that it is necessary to have a new SCM process when working with global software development. However, others ([5], [6]) argue that the synchronization between projects can handle the SCM and make it transparent during development.

To move towards in this question, the objective of this paper is to understand what kind of problems the project teams has faced when working with SCM process in a global software development (GSD) environment, and how these problems have been addressed. To reach this objective, a case study was conducted in a multinational organization with software development centers in Brazil, India and Russia, identifying the difficulties in the SCM process. The results are analyzed and the existing challenges are identified. The results are also showed through lessons learned, that were identified in each project.

Our contribution is the identification of some of the problems and the addressing of the solutions, getting the lessons learned and sharing them inside the organization. This paper has the following structure: section 2 presents the literature review; section 3 describes the research method; section 4 describes the case study developed; section 5 discuss the results found in the case study; section 6 presents the conclusions, future studies and the research limitations.

2. LITERATURE REVIEW

2.1 Global Software Development

Over the last years, software became a vital component in business. Organizations are depending on software as their competitive differential. At the same time, economy has converted national markets in global markets, creating new forms of competition and collaboration [1]. This is part of the globalization efforts currently pervading society, where software project teams have also become geographically distributed in a worldwide scale. This characterizes the Global Software Development (GSD). Several reasons lead to it, which goes beyond demand and cost reduction. Reasons like scale, time-to-market, get capacitated professionals around the world and cultural reasons have moved organizations to GSD ([5], [6]).

Tools and technological solutions have been developed over the last few years to help in the control and coordination of the global development teams working in this kind of environments. Many of these tools are focused in supporting procedures of formal communication such as automated document elaboration, processes and other non-interactive communication channels.

Moreover, Herbsleb et. al [1], and Carmel [5] point out that GSD is one of the biggest business-oriented challenges that the current environment presents under the software development process point of view, including requirements management, software design and the SCM, among others.

Organizations search for competitive ad vantages in terms of cost, quality and flexibility in the area of software development [1], looking for productivity increases as well as risk dilution. Many times the search for these competitive advantages forces organizations to search for external solutions in other countries (offshore developing), generally, these countries provide financial incentives to the companies.

2.2 Software Configuration Management

According to the IEEE Standards [3] and to Berczuk et. al. [2], the purpose of the SCM is to establish and maintain the integrity of the software products throughout the project's software life cycle. Software Configuration Management involves identifying the configuration of the software (i.e., selected software work products and their descriptions) at given points in time, systematically controlling changes to the configuration, and maintaining the integrity and traceability of the configuration throughout the software development life cycle.

The work products placed under software configuration management include the software products that are delivered to the customer (e.g., the software requirements document and the code) and the items that are identified with or required to create these software products (e.g., the compiler). Fujieda et. al. [4] has complementary definition, where the SCM is a set of procedures for tracking and documenting software throughout its lifecycle, to ensure that all changes are recorded and the current state of the software is known and reproducible. This involves creation, and managing the changes in a project plan document.

2.2.1 Software Configuration Management Process

The SCM process defines the sequence of activities that need to be performed in support of the Configuration Management (CM) mechanisms [4]. As with the project management process, the first stage in the SCM process is the planning – identifying those items that need to be under SCM (known as Configuration Items), locations to store them, procedures for change control, etc. Then the process has to be executed. Any SCM process, regardless of whether it uses a tool, requires self-discipline from the project personnel in terms of maintaining versions, storing items in proper locations, and making changes properly. Monitoring the status of the configuration items is therefore important.

3. RESEARCH METHOD

This research is characterized as a study mostly exploratory, since it is based in a theoretical revision and a case study. It is possible to justify the use of qualitative methods since it involves the study of the system development process in its real context, with description and the understanding of the state of art in those situations where practice precedes theory [7].

The case study was conducted in a multinational organization that develops in a global context. The organization is recognized as CMM level 2 since 2002. The objective of the case study was to analyze four distributed projects and to identify problems with the SCM in a global distributed context. At the end of each project, meetings were conducted to also collect lessons learned. This research was organized in four stages (Figure 1).

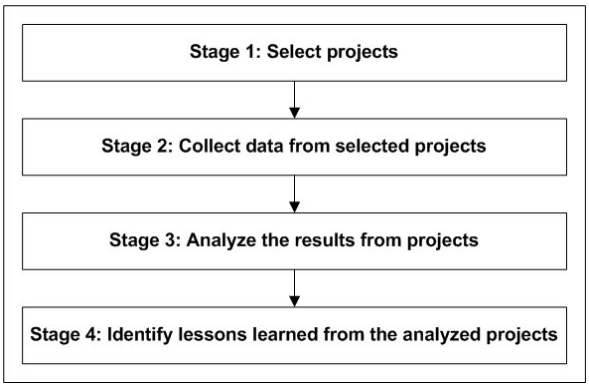

Figure 1. Research Stages

In the stage 1, the objective was to choose some critical projects inside the organization (either maintenance or new projects - developed from the scratch). In the stage 2 data was collected from the analyzed projects in terms of SCM. In the stage 3 the results were analyzed. In the stage 4 the objective was to collect and compose the lessons learned from the analyzed projects, to compose a practical guideline to be used by the project team members for the next projects.

3.1 Characterization of the Organization

The case study was developed in the organization software development center located in *Porto Alegre*, Brazil. This center performs worldwide software development for a multinational organization. Almost all projects are geographically distributed globally since customers and users are located in offices around the world. All customers are internal to the organization. The software development process is based on the MSF (Microsoft Solutions Framework), and on known methodologies such as RUP (Rational Unified Process), and PMI (Project Management Institute). The center studied is recognized as a level 2 organization in the SW-CMM model.

3.2 Characteristics of the Analyzed Projects

The analyzed projects involved more than 40 developers located in 3 different countries – United States, Brazil and India. One project was considered as maintenance project, increasing the functionalities in an existing system. The three other projects were entire new ones in the organization. These projects were analyzed following a timeline of one and a half year. They happened in sequence and the lessons learned found in each one could be applied to the next projects. Because of classified information involved in the projects and to keep confidentiality of the project's purpose, the name of them will not be used as it was. Letters from A to D will be used to reference them.

3.2.1 Project A

Project A was developed in the Java 2 Model View Controller (MVC) architecture, following the structure presented in Geary (2001) [8]. On this architecture, the controller, implemented through a router class, map user inputs (captured in the view) to action classes responsible for the business rules. This mapping is implemented through a resource bundle, a text file containing the action name and its corresponding action class that should be called.

This project was the first to be developed simultaneously in different centers, one in the US, and the other in Brazil. Before that, every project was developed in only one center. The choice of this approach was taken due to the application size and the aggressive delivery date. Working together was the only solution found at that time in order to accomplish the delivery date.

The work breakdown between the two teams was primarily based on the data model knowledge. All data access, implemented through stored procedures, was defined to be done by the US team. Besides that, the US team worked in some of the use cases defined (around 30% of the use cases). The Brazilian team was responsible for the implementation of the other use cases. There were 5 developers in Brazil and 7 developers in US.

Although it was defined that the teams were going to develop code together, each one decided to keep its own SCM environment. Reasons for that were clear: the Brazilian organization had just achieved its CMM level 2. One of the key areas on CMM level 2 is the Software Configuration Management (SCM), which demands a set of well defined and managed processes. This management must be performed with a high level of discipline. SCM activities were periodically audited by the Software Quality Assurance (SQA) team. As the Brazilian SCM processes were brand new, they were also heavy.

On the other hand, the US team, at that time, was not working using a defined process, neither using CMM as a reference. There was a single focal point person to consolidate the changes and synchronize both environments, this person was not aware of the delivery dates and he was allocated in order to perform these SCM tasks only. The result for these differences was that two SCM environments were created: one simple version control for the US team, and other that went further, with life-cycles, baselines, documents and other required capabilities for a CMM level 2 organizations.

3.2.2 Project B

Project B was developed using a proprietary programming language called SEEKER. There is a Human Resources framework developed using this language – much more similar to a SAP system that uses ABAP4 as programming language.

This project was developed simultaneously in three different centers, one in the US, one in Brazil and another one in India. The management team wanted to give some flexibility to the staff team and try to perform a 24 hours development. There were 12 developers in Brazil, 2 developers in the US and 3 in India.

Even with the distributed developers, this project had only 1 main repository and SCM environment, concentrating all the sources, baselines and documents. As the project A, the Brazilian organization already had process for SCM defined according to the CMM model, but neither the US nor the Indian team had the maturity to work in a distributed environment.

In this scenario, all the developers should concentrate their efforts in synchronize their work with one Configuration Management Coordinator (CMC). This is a role that a person took to perform the validation and the integration of the code during the development. It also should provide support for the team during the installation of the product. In this project this role was getting more space in the projects.

A different characteristic is that the CMC was entire involved and synchronized with project needs. He was aware of the target dates and the project purpose and scope.

3.2.3 Project C

Project C used PL/SQL language and the purpose was to create new interfaces to communicate with a human resource system. There were 2 developers in Brazil and 2 developers in the US. The scope of the project were divided in modules and separated between the two sites. Due to an agreement between the teams, they agreed on the use of CMM level 2 compliant processes. But the Brazilian team had more experience which this kind of projects.

This project used a single SCM repository, but involving 2 experienced Configuration Management Controllers to be as focal point. Every kind of work that should be performed should pass by these two persons to validate the artifacts to be uploaded to the SCM tool.

3.2.4 Project D

Project D was developed in the same architecture presented in the project A with similar development team's characteristics: a development team in Brazil and another in US. The Brazilian team was composed by 5 developers, while the US team was composed by 3 developers. The scope of the project were divided in modules and separated between two sites. Just like in project C, the teams were using CMM level 2 compliant processes, but Brazilian organization was working with the processes during more time while US team was preparing their assessment during this project.

Two configuration managers conducted project's SCM coordination, one at each site, supporting the project team members. Experienced SCM coordinators oriented both Configuration Controllers. The SCM environment on this project was shared between the teams.

This project used unique SCM documentation. In one hand that caused a large effort on the planning for defining configuration items and approves the SCM documents. In another hand it

reduced the Configuration Controller effort during development phase acting on Quality Audit's non-compliances and on rework related to re-planning SCM activities and baselines.

4. CASE STUDY RESULTS

In order to organize the analysis, Table 1 shows some critical points identified in the projects. For the country identification, we have used Brazil as "BR", the Unites States as "US", and India as "IN". It is important to notice that the projects were developed in sequence, one after the other.

Table 1. Critical data collected from the analyzed projects

Characteristics of projects				
Project	Countries involved	# of SCM focal points	# of developers	Use of distributed environment
A	BR, US	1	12	NO
B	BR, IN, US	1	17	YES
C	BR, US	2	4	YES
D	BR, US	2	8	YES

We can see that the number of SCM focal points increased from project A to project D. The tendency in using distributed enviroments could also be noted only in project A, showing that this approach was not to effective in the first distributed project. The most complex project, in terms of SCM, was project B because the high number of developers from different cultures, using different processes. In each project, an extensive analysis was conducted to identify the problems and lessons learned. The commom lessons learned will also be higlighted in each project.

4.1.1 Project A

On the Project A, as discussed in the previous section, the work breakdown between the American and Brazilian teams was based on two points: all data access would be performed by the American team (which had better knowledge on the data model), and the use cases would be divided between the two teams. This breakdown caused problems identified in the middle of the development phase: the American team, responsible for the stored procedures, got over allocated and delayed the stored procedures required by the Brazilian team. That impacted the Brazilian team a lot, who depended on the American team in order to finish their use cases. The project schedule was affected because of that. For that reason, **the work breakdown in distributed projects should minimize dependencies between geographically distributed teams**.

The decision to keep distinct SCM environments for each team brought together several consequences. The first was the need for synchronization on the SCM environments. In project A, as the SCM tools were different, this had to be performed manually, a task that took from the configuration controller 3 to 4 hours each week.

Some configuration items were updated by the two teams: the resource bundle file, for example. This obligated the configuration controller to perform manual file merges on the resource bundles, a task subject to errors. In that context, **distributed development projects should work with only one instance of SCM environment**.

4.1.2 Project B

Project B used a single instance of SCM environment, and that showed later to be a crucial decision for the project success. Processes were previously negotiated between both teams, thus avoiding synchronization and build problems.

At the end of the development phase, and because of the project size and the number of people testing the application, it was decided to perform two builds every day. This number of builds required one responsible person with great application and technology knowledge. Builds were scheduled in predefined times. Although builds were frequent, the number of errors caused because of builds was very low, around 3% of the total number of errors found in the project. **Even with centralized SCM environments, the team should define one build coordinator with great experience on the application and technology in place**.

Although SCM process were previously negotiated and agreed between the American and Brazilian teams, it was noted that some fundamental concepts in SCM were misunderstood or misused. The probable reason for that was that the American team was starting its work towards CMM level 2, while the Brazilian team had already got this certification level, and was, consequently, more experienced in the SCM processes.

An example of such situation was the baseline concept. The baseline should be used as the place from where the build coordinator extracts the configuration items required for the build. This concept defines and directs what and how the configuration items should be created: all items should have high cohesion, and do not depend on other items that are not under configuration management. The contents of a baseline should be enough to reproduce an application environment at any time, now or years in the future.

In project B it had such situation with database scripts, that although were under configuration management, they were supposed to update an object that was not the database itself. There are ways to avoid this kind of problems in scripts: do not use updates, remove all records before inserting into a table, etc. They were actually ways that allowed the creation of the application environment from scratch, at that time. In other words, **putting all configuration items required for a build under configuration management is a good approach**.

It was also noted that that baselines were not enough to rebuild an application environment from scratch. It was necessary to use other resources that were not under configuration management (database backups, for example). Hence, **establish and clarify all main concepts on SCM discipline, before actually starting development**, can be an approach to avoid lack of understanding. Reviewing and checking the configuration items is a good solution also, because it can prevent missing files that aren't under SCM.

4.1.3 Project C

Project C was characterized by the weak engagement about the SCM processes. That caused some misunderstandings between the American and Brazilian teams. The SCM on this project was handled by two configuration controllers, one in the American team, and the other in the Brazilian team, but responsibilities of

each one were not previously defined and communicated to the project stakeholders.

An example of direct consequence on this was the high number of non-compliances found: the SQA (Software Quality Assurance) team of each country didn't know what the scope to be evaluated was, and that caused a lot of problems that could be easily avoided with a better engagement at the beginning of the project. **Even with experienced teams in distributed development, the SCM engagement at the beginning of the project should be prioritized**.

Another problem faced by the development team on project C was the lack of baselines planning. The baselines were requested on demand, and sometimes could not be handled by configuration controller because of his allocation on other tasks. This caused delays in some planned deliveries. For that reason, is good to always **plan baselines and document them in the project's SCM plan, as soon as possible**.

4.1.4 Project D

Project D was the last project developed in timeline from the projects shown in this work. With that this project didn't experienced many of the problems of the projects A, B and C, but some of them were still noticed on this project.

The most noticed problem experienced in the past was the dependency between the modules developed by the American and Brazilian teams. Even with a less impact than the project A, modules dependency caused some rework and idle time for the developers.

Mainly that was caused by changes on the scope of each team, part of the scope of a team passed to the other due to some delay. With that a module were divided between the teams and the problem with the dependency appeared once again. **The re-planed activities due to scope floating across teams should take in place. The analysis should include dependency verification on the module being floated against the other modules**.

5. LESSONS LEARNED

After the analysis of these four projects, it can be conclude that to manage the configuration in a GSD context can become an arduous task if the process will not be well defined and if the teams will not be previously prepared to work in this scenario.

What was perceived here is that all the work involving the CMM Model level 2 project in the Brazilian unit collaborated in a big scale to minimize some problems found in this scenario. The definition of a SCM process based on the CMM model brought excellent results related to the distributed environments problems. Also, the teams were able to standardize all the work and to converge in a common understanding about the best approach to develop both projects. So, as a conclusion, many of the efforts spent in the CMM Model level 2 project contributed to minimize problems in terms of SCM process definition.

Looking at the timeline of the projects presented on this paper, some considerations can be identified for the evolution of the SCM activities in distributed projects executed by this organization. It was noted that some of the lessons learned in prior projects are in fact being applied in the most recent projects. An example of this is the unique SCM environment. Project A was the last distributed project executed in two distinct SCM environments. The overhead caused by this multiple environment can make the distributed development impossible, as the project size increases.

Other lessons learned were forgotten as time goes by. The dependency on tasks executed by the American and Brazilian teams, identified on project A, is back on project D, although it didn't happened on projects B and C. For some reason, the teams are recurring to an alternative that will cause more problems in the near future. Those reasons must be better analyzed.

There are also other lessons learned that in fact were not assimilated by the project teams. An example of that is the problem in fundamental concepts in SCM, like the baseline concept, identified on project B. Although the problem was raised, it still is not considered as a critical factor in the future projects. Project teams did not understand yet the importance of such concept, and the benefits that they could have with its correct utilization.

It was noted also a tendency on the teams to relax on engagements, as times goes by. The SCM engagement observed on project C was very poor, and that caused a lot of SQA issues. Maybe the experience got in projects A and B brought together a false sense of power on the teams, which caused this weak engagement. A list of the problems and lesson learned identified in the projects described in Table 2.

Table 2. Lessons Learned

No.	Lesson Learned	Projects
#1	The work breakdown in distributed projects should minimize dependencies between geographically distributed teams.	A
#2	Distributed development projects should work with only one instance of SCM environment.	A
#3	Put all configuration items required for a build under configuration management is a good approach.	B, C
#4	Distributed development projects with centralized SCM environments should define one build coordinator.	B, C
#5	Establish and clarify all main concepts on SCM discipline, before actually starting the development, is a good approach.	B
#6	Even with experienced teams in distributed development, the SCM engagement in the beginning should be prioritized.	C
#7	Always plan baselines and document them in the project's SCM plan, as soon as possible.	C, D
#8	The re-planned activities due to scope floating across teams should take in place.	D

Lesson Learned #1: If possible, each team should work in their modules without any dependency. No matter how integrated the distributed teams are communication will always be bureaucratic and expensive. This is related with the work breakdown, causing dependency among distributed teams, creating a great impact in the development.

Lesson Learned #2: Both teams should agree in a common management process. Distinct SCM environments caused overhead on CM work and activities.

Lesson Learned #3: All files related to build (source codes and build files) should be stored in a global configuration environment. Even training and end-user documents can be stored in the same environment.

Lesson Learned #4: The configuration manager should have great experience on the application and technology in place, so, the build will be more efficient. This was because the high number of required builds throughout the project development.

Lesson Learned #5: Some concepts are essential for the understanding of what needs to be under configuration management, and its contents. It was noted that the fundamental concepts in SCM weren't completely understood by project team members.

Lesson Learned #6: Focus on the set of SCM processes, responsibilities of each team member and communication. This can avoid the weak engagement on SCM at the beginning of the project.

Lesson Learned #7: Avoid requesting baselines on demand; otherwise the deliveries can be delayed. The lack of planned baselines is a problem root cause.

Lesson Learned #8: The analysis should include dependency verification on the module being floated against the other modules. This can predict the problems with configuration management activities.

6. CONCLUSIONS

The SCM has a critical role in software development process. The configuration artifacts are used in all subsequent phases of software planning and developing. Developing, testing, deployment and installation are made based on the software configuration. There are several difficulties when trying to synchronize the SCM activities between teams. Most of those difficulties are increased when software development teams are distributed, in fact, some new difficulties can appear.

Considering the growing adoption of the GSD, there are few studies about the impact it has in the SCM process and tasks. In these studies, the technical aspects aren't considered in detail. It is clearly necessary processes, patterns and tools to address difficulties cause by team distribution in terms of SCM.

This paper advances the knowledge in the GSD area when identifying some important characteristics of the SCM in a global environment, in parallel with a CMM Model level 2 certification process organizational analysis, specific in a CMM key process area. As result, many important issues were identified and many lessons were learned.

This study enables a better understanding of the GSD area and the relationship between the project team and users related to the SCM. Due to the small number of case studies, the results cannot be generalized. In this phase, we can adopt the analytical generalization principle, proposed by Yin [7]. Also, it is important to notice that this study was not considered an analysis of the reasons than can take an organization to adopt strategies of distribution, nor the software development process by itself.

The intention is to run this study again to collect more empirical data, bringing more accuracy to the results. Also, new researches will explore some alternatives and solutions related to the GSD process identified, considering all difficulties and critical success factors like culture, communication, coordination, trust and cooperation focused in the SCM process. In the same way, another comparison can be done between the four projects analyzed, in order to understand the way that each team did his work, considering the distribution level and the team's profile.

As the main contributions of this study, we can highlight the lessons learned and the main advantages in having the lessons learned applied in each subsequent project. Moreover, continuous software process improvement based on a quality model can be very important to succeed in GSD environments.

7. ACKNOWLEDGMENTS

This study was developed by the Research Group on Globally Distributed Software Development of the PDTI program, financed by Dell Computers of Brazil Ltd. with resources of Law 8.248/91.

8. REFERENCES

[1] Herbsleb, J. D. and Moitra, D. (2001). Guest editors' introduction: Global software development. *IEEE Software*, 18(2):16-20.

[2] Berczuk, S. P., and Appleton , B. (2002). *Software Configuration Management Patterns: Effective Teamwork, Practical Integration*, Addison Wesley.

[3] IEEE Standards (1999) – Software Engineering – Volume Two – Process Standards.

[4] Fujieda , K., and Ochimizu, K. (2003). Investigation of Repository Reprecation Models in Globally Distributed Configuration Management. In *Proc. of the Workshop on Global Software Development at ICSE*.

[5] Carmel , E. (1999). *Global Software Teams – Collaborating Across Borders and Time-Zones*. Prentice Hall, USA.

[6] Herbsleb, J. D., Mockus, A. Finholt, T. A., and Grinter, R. E. (2001). An Empirical Study of Global Software Development: Distance and Speed. In *Proc. of the 23rd International Conference on Software Engineering*, 81-90.

[7] Yin, R. (2001). *Case Study: Planning and Methods*. Sage, USA.

[8] Geary, D. (2001). *Advanced Java Server Pages*. Prentice Hall. Sun Microsystems Inc, USA.