2012 IEEE Seventh International Conference on Global Software Engineering

# Configuration Management Stories from the Distributed Software Development Trenches

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

Jan Magnusson
Sony Mobile Communications
Lund, Sweden
Jan.Magnusson@sonymobile.com

Christian Pendleton
Softhouse Consulting
Malmö, Sweden
christian.pendleton@softhouse.com

***Abstract*—Distributed projects are generally recognized as being more complex and adding a number of new challenges to project management. Configuration management (CM) can be considered the infrastructure of all types of project being they co-located or distributed and lack of CM or badly implemented CM will hurt any type of project. In this paper, we take a closer look at the role of CM in distributed projects – where can standard CM techniques help, how can they be implemented, and what special challenges does distribution pose. We do that by looking at general and CM-specific challenges from literature on global software development and discuss those in the light of our experience as CM practitioners on different distributed industrial projects. Some challenges in distributed development can be solved or alleviated by CM techniques, for other challenges CM has to be implemented differently – and some challenges are challenges to CM too.**

***Keywords-configuration management; distributed development; challenges; lessons learned; experience***

## I. INTRODUCTION

In contrast to co-located projects, distributed development (DD) is recognized as making projects more complex and adding new challenges [12]. Configuration management (CM) can provide the infrastructure for any type of project whether co-located or distributed and CM is considered a fundamental capability that has to be in place for an organization to progress to level 2 of the Capability Maturity Model [8].

As CM practitioners we have worked on many different projects over the years, both distributed and co-located. We have noticed many similarities from our special CM point of view but also some differences. We wanted to understand the challenges of DD and know more about how that is different from the well-known co-located CM setup. However, we found very little literature that could give a general overview of CM on DD projects [2], [9]. Most literature seems to focus narrowly on one single problem and/or solution and leave you with a very fragmented picture of in which ways CM can contribute on a DD project.

Furthermore, what we read in literature did not always correspond to our own experience and understanding of what CM is and what CM does. Sometimes issues were brought forward that we could not relate to as a CM responsibility. Sometimes issues were promoted as distributed challenges when they, in our opinion and experience, were "just" lack of well understood CM. So we had to work really hard to figure out if we had gotten something wrong or misunderstood the message – or if we had something new to contribute to CM on DD projects.

We want to share our experience and insights from this work. To do that in a systematic and structured way, we started from general and CM-related challenges in DD drawn from other people's literature reviews. We carefully reviewed the general challenges to identify those that can be considered to be CM-related, either because they belong to the CM area of responsibility or because CM could provide a solution to the challenge. To that we added the CM-related challenges that others have identified. This list was then discussed and compared to experience from our case studies to arrive at a better understanding of what are CM-related challenges and what are not – and which CM-related challenges are already solved and which remain open.

We want to share our lessons learned primarily with other CM colleagues, who might struggle with distributed projects – but also with project managers and others, who sometimes are unaware of what CM can offer to their project. Finally, we also want to share with researchers, so they can come up with solutions for situations where we have failed.

In the following, we first give a short introduction to the most basic concepts and principles of CM, then we review literature for challenges in DD that are related to CM, we give brief descriptions of our cases, before we discuss our experience from the cases in relation to established challenges and issues from literature and standard CM theory and practice. Finally, we draw our conclusions.

## II. CONFIGURATION MANAGEMENT

In this section, we give a short introduction to the most important concepts and principles in CM. It will give a slightly alternative picture of CM to the CM-knowledgeable readers and will make it easier for the CM-uninitiated reader to follow our subsequent discussions and reasoning.

### A. Traditional Configuration Management

Traditionally CM is considered to consist of four activities: identification, control, status accounting and audit [7].

978-0-7695-4787-9/12 $26.00 © 2012 IEEE
DOI 10.1109/ICGSE.2012.32

The purpose of *Configuration Identification* is to make sure that all important parts of a project are identified and put under configuration control. These *Configuration Items* are described and defined and it is decided how Configuration Items should be named and structured to allow easy retrieval and recognition and identification. Defined groups of Configuration Items can make up *configurations* (like a requirements specification) and Configuration Items can be related to other Configuration Items to give *traceability* (like tracing a requirement to its tests and implementation).

In *Configuration Control* focus is on managing changes to configurations. Once a given configuration is stable a *baseline* is defined for that configuration. The only way to make changes to a baseline is to create a *Change Request* (problem report, deviations, waivers are synonyms) and take it through the *change management process*. The central part of that is the *Change Control Board* that makes decisions about whether to accept or reject a Change Request based on information provided and that subsequently follows the status of the Change Request through to its closure.

*Configuration Status Accounting* is the activity that can provide all sorts of *information* to all sorts of people. Traditionally it is looked at as producing printed paper reports of information about the status of the change management process for the project manager with regular intervals. However, more generally the status accounting activity can make available also more dynamic information (like "who is changing this file") through other types of media (like a wiki) and for other types of "customer" (like testers or developers).

Finally, *Configuration Audit* has the purpose of making sure that we are ready to deliver what has been promised and is done in a more formalized way prior to release. The *Functional Configuration Audit* is a sanity check for whether the prescribed change management process has been followed – have all accepted Change Requests gone through all steps of the process to end up in the "closed" state. The *Physical Configuration Audit* checks whether all physical parts (like memory card, user manual or help files) of the product are there and correspond to their description.

### B. Team-oriented Configuration Management

Internally in smaller teams people had struggled with the day-to-day coordination of their parallel work and gradually "invented" CM processes and tools to help them out.

Wayne Babich [1] very eloquently identified three fundamental problems that he had seen happen in the coordination of individual people's work in a team: shared data, simultaneous update and double maintenance. Problems that we can never hope to eliminate, but that we can manage by the use of good processes or tools. The *shared data* problem is the situation where a problem is caused by the changes of other people – changes that we are not aware of. The concept of a workspace in version control tools will isolate us from other people's changes. However, when we synchronize our workspace with the repository we invite in the shared data problem. The *simultaneous update* problem is when someone accidently overwrites and removes someone else's change. Version control tools will not allow us to commit a change to the repository if someone else has already committed a change and thus avoids the danger of overwriting. However, when we resolve merge conflicts there is the possibility that we accidently remove (parts of) other people's changes. The *double maintenance* problem happens when we copy something – and make changes to one of the copies. In order to keep the two copies identical we have to make the exact same change in the other copy too. A merge tool will help us do that automatically – if we know that the copy exists. However, most merge tools only work for lines of text.

Peter Feiler [3] distilled the work models of version control tools. For the synchronization part early tools provided locking mechanisms to stop parallel work on the same components, but gradually moved towards a more relaxed model that allowed parallel work since there was tool support to merge parallel changes. The transaction model changed from a very simplistic model where people were focused on single files committed one at a time towards the concept of logical changes where a set of changes was committed in one atomic operation. To create configurations (e. g. when populating a workspace) there are two models. The composition model creates a new configuration from a system model that gave the general architecture on which a selection rule is applied to pick one version for each node in the system model. The change set model creates a configuration from a baseline on which is applied a set of changes. Some tools support only very limited parts of one configuration model, other tools fully support both models.

## III. Literature Review

We did not want to create a list of CM-related challenges in DD based exclusively on our own experience and cases, so we needed to review literature for what others might have found. Much has been written about general challenges in DD and many have written about different aspects of CM in DD. Fortunately there are also some existing literature reviews, so we did not have to go through oceans of papers. We have based our review of general challenges on [12] and [6]. There are many others like [5], [10] and [4], but we did not find that they added anything substantial to the primary literature reviews we use, which confirmed our impression that we were on solid ground. For the CM-related challenges there is very little. We have only been able to find [2] and [9] that form the literature basis for our subsequent discussions.

da Silva et al. [12] collect and systematize reported knowledge in terms of what are the difficulties in DD projects. Though the review is based on a much larger pool of literature, they extract information from a final selection of 54 papers. They come up with a list of 30 general challenges of which the first five accounts for almost half of the found mentionings. We find cultural differences to be outside the scope of CM, whereas *coordination* is strongly related and *effective communication*, *time zone differences* and *trust* are weakly related. It seems like they focus more on trust on people, whereas we have more focus on trust on code (or artefacts in general). From the remaining 25 challenges, some (like need of office space) bear no relation to CM, while others (*physical distance*, *task allocation*) are

weakly related and quite a few (*different knowledge levels or knowledge transfer*, *tracking and control*, *cooperation*, *knowledge management*, *scope and change management*, *differences in technologies used*, *synchronization work between distributed sites*) are strongly related to CM – either because they pose problems for CM or because CM can help manage the challenge. They also find 31 best practices that are used to manage the challenges. One of these is to deploy and use a configuration management system and since they talk about it in general terms we take it to include all aspects of CM as laid out in section 2. Therefore it surprises us that when they map best practices and challenges, CM is only mapped to the "effective communication" and the "trust" challenges. We believe that CM can help deal with many other challenges too. When they map challenges and tools, CM (which is then a change management system) is mapped to the "cooperation" challenge and the "scope and change management" challenge. Again, CM can help with these challenges, but we also believe that it can do more. However, we are rather puzzled that they map the CM tool to the "multiple communication modes" best practice.

Jiménez et al. [6] synthesize the findings of their literature review (based on 78 primary studies) of challenges in DD into 10 different areas. Since their areas are more general and wide in nature than the challenges of [12], we find that most areas are in some way related to CM. We consider the following areas as strongly related to CM: *Software configuration management* (though they seem to take that as source code control and awareness), *Knowledge management*, *Coordination and Collaboration*. The areas we consider as more weakly related to CM are: *Communication*, *Group awareness*, *Project and process management*, and *Risk management*. Whereas we do not see any particular relations between CM and Process support or CM and Quality and measurement.

Pilatti et al. [9] analyzed CM in a DD environment to identify the main challenges. They studied four distributed projects and found 8 different issues that they relate to CM. We agree that all of their issues are matters of CM. However, most of them (like "always plan baselines and document them in the project's SCM plan as soon as possible") are important CM issues also for co-located projects and as such the "real" problem can be considered lack of using known CM concepts and principles. There are two issues (the work breakdown in distributed projects should minimize dependencies between geographically distributed groups, distributed development projects should work with only one instance of SCM environment) that, even though they are also known from co-located projects, may be more critical on a DD project.

Fauzi et al. [2] experienced a lack of attention to technical areas of DD research and carried out a systematic mapping of literature to identify CM issues in DD based on a final selection of 24 primary papers. They distinguish 13 CM-related problems faced by developers in DD:

- (P1): Dispersed software teams do not get information on what other teams are doing
- (P2): Difficult to know the traceability of each module
- (P3): The definition of modifications or problems to be handled is unclear
- (P4): Dependency
- (P5): Delay and increased time required to complete change requests
- (P6): Working in different SCM environments
- (P7): Change requests are handled at various levels in the project
- (P8): Lack of a planned baseline
- (P9): Lack of coding standards
- (P10): Code ownership
- (P11): Unclear flow of development
- (P12): Tool selection
- (P13): Artefacts with different versions and content at each site

Just as for the issues in [9], we would be hard pressed as CM people to feel particular responsibility for some of these issues (like "code ownership"). Likewise we consider many of the issues (like "lack of a planned baseline") as plain negligence of well-known CM concepts and principles. Others ("working in different CM environments"), however, are things that CM will have to deal with on a DD project. This is further discussed in section V.

## IV. Case Descriptions

In this section, we give short descriptions of the cases we use in the discussion of our experience in the following section. Not all cases are explicitly referenced in the discussions, but they all have contributed to the experience we draw on.

**Case I:** Company A is a large development company (>150 developers at this unit), where distributed development is quite common. They hired consultants from company B, which is distributed on two sites within one country with the consultants working from their "home" office. Four development teams at company B are working remotely (three at one site and one team in a separate, all in all about 30 developers). The deliverables from company B consists of source code files. To solve the distribution situation, company A offers a remote desktop connection solution, making the consultants connect from their local offices to terminal servers inside company A. The remote desktop solution gives access to the whole development environment at the company including tools for version control, code review, system integration, test etc.

**Case IV:** A division of an international consumer electronics developer spanning several sites across different continents. The major offices are located in Tokyo, Beijing, Lund (Sweden) and San Francisco. The software development organization in this division numbers 1000+ people. Teams are organized around products, components or features as the situation dictates. Product and component responsibility will be located at one site, but development activities can be assigned to teams in other sites. Software development is regularly outsourced and off-the-shelf components can also be included in finished products. The development environment inside the company is very homogeneous with centralized build resources and tool

management. For outsourced teams, an SDK is provided in most cases, but if so required, a remote desktop solution is also offered.

**Case VI (anti-case):** This case is the complete opposite of the other cases as it describes an extremely co-located setup where everyone on the project team is in the same room at the same time. The purpose of this case is to uncover CM solutions that are implemented differently on a DD project and CM solutions that are simply "implemented by communication" on a co-located project. A group of 8-10 students has to produce an application to manage motorcycle competitions. They work closely together with a customer, develop following eXtreme Programming and are being coached by two older students. The team has its own room where all project activities take place and everyone works at the same time. Each iteration starts with a two-hour planning game and ends with an eight-hour programming session. In between up to four hours of individual work per student can be spent on spikes (e. g. looking through the code for bad smells/missing unit tests, looking into how to use Ant for the release, baking cake – or whatever they feel is needed in preparation for the programming session). The project runs for 6 iterations and the final (and fourth!) release is complete with applications, user manuals, source code, and technical documentation.

## V. DISCUSSION

In the following, we discuss a small selection of topics drawn from our experience with the cases described in the previous section. Discussions will also relate to CM concepts and principles and/or previous findings from literature.

**Code ownership in a distributed organisation (from case IV):** Large systems have the drawback that each developer cannot be fully cross functional in the sense that he/she is familiar with all parts of the code. Many organisations solve this by appointing ownership of code modules. Working feature oriented with such a system will sooner or later create a situation where the developers will have to change code that is "owned" by someone else since some features is naturally spanning over many modules. The owner of the changed code naturally will want to know when someone else is committing changes to his/her module and in many cases also review the changes before they are merged to the common code base. This functionality can be offered by good version control, supporting feature branches, in combination with a process and a tool for code reviews.

**Dispersed software teams do not get information on what other teams are doing (from case IV and P1 [2]):** In a distributed team setup, information about ongoing activities will not naturally be passed from developer to developer across sites. If the only interactions with remote developers happen through the code repository when artefacts are retrieved or stored or the repository is queried, developers will be quite reliant on real time communication in order to avoid or resolve conflicts. The strategies suggested by [11], namely, well defined tasks and exclusive areas of responsibility, is only valid if the architecture of the software worked on is such, that there are well-defined components with a clear and shared understanding in the organization of their scope and functionality and when adding two pieces of seemingly unrelated functionality, the probability for them to be dependent is low. And even if those two architectural requirements are fulfilled, the result is not mainly improved awareness of remote developers activities, but instead a way to reduce the risk that the communication deficit resulting from a large and/or distributed development organization affects ongoing development. Another commonly used strategy to tackle the risk of unnoticed dependencies interfering with ongoing work, is simply to limit the amount of work-in-progress, where continuous integration would be an example. Strategies aimed at improving awareness should encourage sharing rather than isolation. Introducing a review process, whereby all changes must pass this process before being introduced into the main development code line, has greatly improved the ability for developers in the organization in Case IV to share ongoing work with peers as well as given them clear benefits by doing so.

**Control over a distributed environment (from case I and P6 [2]):** Working distributed, there is always a risk that the development environment starts to diverge between the sites. This may affect efficiency in the development work in different ways but it can also impact traceability through the systems and even how software is built, making it hard to reproduce binaries at a different site. CM control over the development tools is essential in these cases.

**Access control in a distributed environment (from case I and IV):** Since operative CM is involved in the development environment and the version control tool handles access to the source code, CM often gets involved in the access control setup. When working distributed, there is a risk that different sites handles access control differently. Every development network should have a strategy for access control and tools or infrastructure that supports this strategy. In cases I and IV, remote desktop solutions automatically gave a consistent environment with the proper access rights applied without requiring any extra work or solutions. In case IV, access control is also governed by tools offering support for assigning code access permissions.

**Commit shouting (from case VI):** The fact that everyone is potentially working on the whole system in parallel means that the team has created the double (actually multiple) maintenance problem from [1]. To handle that they try to integrate as often as possible to keep the "integration effort" as small as possible. In the beginning they update continuously, but soon they discover that there is nothing new in the repository to integrate and these "idle" updates become noise in their work. Most teams then adopt the practice to shout out "commit" whenever someone commits something to the repository – after which all the others know that now there is something new and that they should update and synchronize as soon as possible/convenient. The technical CM solution to this coordination problem does not care whether the team is co-located or distributed – however, the implementation of the shouting has to be different.

**Dependency and Delay (P4, P5 [2] and case IV):** Given a certain architecture and development process, there is a resulting probability that two random ongoing changes will depend on each other in one way or another. As software

grows in size and complexity, so normally does the number of developers involved and the number of changes made to the software in a certain time period. Often, the architecture and processes will adapt in order to reduce the probability that two changes will affect each other, but this probability will never be zero. This means that whatever process and architecture there is, sometimes such dependencies will need to be dealt with. In a large and complex organization this work faces many challenges. There must exist a way to discover ongoing changes and that they are dependent. If developer lacks the knowledge or experience in a certain are where changes are required, it must be possible to find relevant experts. You must be able to communicate with developers of dependent changes or experts and it must be possible to get authorization and acceptance for the changes needed from both managers and engineers as appropriate. When the changes are completed, it must also be possible to introduce them into the system in such a way that it does not break. CM processes and tools can designed to help out in all of those areas, but time zone differences in the magnitude of a working day, will result in synchronous communication of any kind becoming a major problem. Often this leads to that inefficient modes of communication are used and causes higher than normal error rates, as well as request-response times scaling days instead of minutes or hours. In case IV, the means to reduce time zone induced lead times have so far been very costly and involve duplication of resources in order to have expertise and authority available at all times when development work is ongoing.

**Distributed version control tools are better suited (case VI):** In this case there was the belief from two of the teams that a shift from a centralized to a distributed version control tool would ease the coordination task. It turned out that they ended up needing a more centralized and controlled integration process than the other teams – most probably because their belief had lured them into integrating less often. In fact, the only basic difference between the two paradigms is that the workspace is a repository on its own – with possibility for version control – in the distributed paradigm. The integration and coordination of contributions does not become any easier. However, in project setups with network problems distributed tools could work better.

## VI. Conclusions

Many issues found in DD stem from communication problems. CM can help alleviate some of the issues by providing a framework for communication through the implementation of standards for identification, status accounting and decision-making. Working in a small, co-located team, a common nomenclature will normally evolve through personal interactions. Status accounting can happen by "shouting" and data required in order to make informed decisions can be retrieved by simply arranging a meeting with the whole team or talking directly to the right person.

As soon as the formation of these informal practises is hindered, either by project size, complexity or distribution, the ability to perform basic tasks crumble unless a more structured approach is introduced. In many cases, this structured approach is simply implementing current CM best practices, without particular consideration to if the work is distributed or not. The real issue seems to be to which extent informal procedures are prevented to form and distribution is only one such roadblock. One important aspect though, is that the CM procedures introduced, must be organization wide in order to serve as a communication framework. Heterogeneous environments and procedures will complicate the matter, but is left for later examination.

There are some additional issues that are limited to DD projects. Poor infrastructure, with limited bandwidth, network instability, high latency, etc, can prevent the implementation of efficient formal CM practices or introduce considerable delays and queues. Time zone differences is another problem that by its very nature will also introduce delays that hinder communication, regardless of how much of CM best practices that are introduced.

## Acknowledgement

We would like to thank Marc Girod, Ericsson, Ireland, Ulf Steen, ABB, Sweden and Torben Poulsen, Thy:data, Denmark for comments, ideas and provision of cases.

## References

[1] W. A. Babich: "Software Configuration Management – Coordination for Team Productivity", Addison-Wesley Publishing Company, 1986.

[2] S. S. M. Fauzi, P. L. Bannerman, and M. Staples: "Software Configuration Management: A Systematic Map", in Proceedings of the 17th Asia Pacific Software Engineering Conference, Sydney, Australia, November 30 – December 3, 2010.

[3] P. H. Feiler: "Configuration Management Models in Commercial Environments", Technical Report, CMU/SEI-91-TR-7, Carnegie-Mellon University, Pennsylvania, March 1991.

[4] S. ul Haq, M. Raza, A. Zia, M. Khan: "Issues in Global Software Development: A Critical Review", Journal of Software Engineering and Applications, Number 4, 2011.

[5] J. D. Herbsleb: "Global Software Engineering: The Future of Socio-technical Coordination", in Proceedings of Future of Software Engineering, Minneapolis, Minnesota, May 23-25, 2007.

[6] M. Jiménez, M. Piattini, and A. Vizcaíno: "Challenges and Improvements in Distributed Software Development: A Systematic Review", Advances in Software Engineering, Volume 2009, 2009.

[7] A. Leon: "Software Configuration Management Handbook", (second edition), Artech House, 2005.

[8] M. C. Paulk, C. V. Weber, B. Curtis, M. B. Chrissis: "The Capability Maturity Model: Guidelines for Improving the Software Process", Addison-Wesley Publishing Company, 1995.

[9] L. Pilatti, J. Audy, R. Prikladnicki: "Software Configuration Management over a Global Software Development Environment: Lessons Learned from a Case Study", in Proceedings of the International Workshop on Global software development for the practitioner, Shanghai, China, May 23, 2006.

[10] C. Prause, R. Reiners, S. Dencheva: "Empirical Study of Tool Support in Highly Distributed Research Projects", in Proceedings of the Fifth ICGSE, Princeton, New Jersey, August 23-26, 2010.

[11] R. Sangwan, N. Mullick, M. Bass, D.J. Paulish, and J. Kazmeier: "Global Software Development Handbook", Auerbach Publications Taylor & Francis Group, 2006.

[12] F. Q. B. da Silva, C. Costa, A. C. C. França, and R. Prikladnicki: "Challenges and Solutions in Distributed Software Development Project Management: A Systematic Literature Review", in Proceedings of the 5th International Conference on Global Software Engineering, Princeton, New Jersey, August 23-26, 2010.