

Version Control in Distributed Software Development: a Systematic Mapping Study

Catarina Costa^{1,2}

¹Statistics and Mathematics Center
Federal University of Acre
Rio Branco - AC, Brazil
catarina@ufac.br

Leonardo Murta²

²Computing Institute
Fluminense Federal University
Niterói - RJ, Brazil
leomurta@ic.uff.br

Abstract— Along the last decade, many companies started using Distributed Software Development (DSD). The distribution of the software development teams over the globe has become almost a rule in large companies. However, in this context, new problems arise, which mainly involve the physical and temporal distance among the participants. Some studies show that deploying a version control system to alleviate this problem is a big challenge for distributed teams. This paper presents a systematic mapping study about works about version control that focus on DSD. We found 29 studies related to DSD version control, published between 2002 and 2012. Using the systematically extracted data from these works, we present challenges, tools, and other solutions proposed to version control in DSD. These results can support practitioners and researchers to better understand and overcome the challenges related to DSD version control, and devise more effective solutions to improve version control in a distributed setting.

Keywords- *version control; distributed software development; configuration management; systematic mapping study.*

I. INTRODUCTION

The Software Engineering process has been subject to some globalization-related changes. IT experts all over the world have witnessed the growth of Distributed Software Development (DSD) along with its increasing popularity, which is defined by Carmel [1] as “a software development model whose software development team is physically apart”.

Countries like India, Brazil, and Ireland, among others, offer excellent resources and fiscal incentives to develop software [2]. DSD gained momentum as it proposed spectacular benefits, such as work cost reduction, skilled development team, flexibility to allow in-house staffing, and quickly adaptation to volatile business needs [3].

However, it also attracted attention due to the complexity and challenges related to globally distributed development teams. Some studies reported that the DSD scenario enlarges ordinary software development obstacles and adds more challenges such as time zone, geographical, and socio-cultural differences [1], [3], [4], [5], [6], [7].

As managing the consistency and concurrency among project artifacts is an issue, Configuration Management (CM) earns special attention [2], [4], [7], [8]. CM intends to control the software evolution, especially with the help of Version Control Systems (VCS). It not only aids evolution

control, but also supports parallel development, which is a usual situation in DSD environments.

CM can provide infrastructure for any type of project, whether co-located or distributed. However, the geographical distance heightens the challenges faced by any development team, such as communication among team members, full understanding of the project, and system integration. This scenario makes the work almost infeasible without a CM tool, such as Subversion, Git, or Mercurial. Although some of these and other VCS support parallel development, most of them were not specifically designed to deal with the DSD idiosyncrasies.

VCS allow engineers to work on software artifacts independently and with reduced planning and coordination because they automatically merge changes made in those artifacts and detect conflicting modifications. However, they do not detect conflicts until the engineers check-in changes, at which point unnecessary effort may have already occurred, for example. Furthermore, conflicts may be more difficult and time-consuming to resolve at this later stage [9].

In addition, some CM tools have features that make them more or less prepared to deal with the DSD characteristics. Tools like CVS and Subversion adopt a centralized topology. Both follow the client-server model, use a single central server that hosts the project’s metadata, and provide to developers a limited amount of data that represent specific versions. On the other hand, Git and Mercurial adopt a distributed topology. They operate in a peer-to-peer manner. Every copy of a project contains all the project’s history and metadata [10].

An important topic to be considered is which topology is more widely used and/or proposed by developers and researchers in the context of DSD, aside from the most utilized VCS tool. According to O’Sullivan [10], if agility, innovation, and remote work are essential for a certain project, distributed VCS are more capable of meeting this project’s requirements. On the other hand, for organizations concerned with data security, centralized VCS are definitely more appropriate.

Only a few primary and secondary researches on the subject have been conducted. Fauzi *et al.* [7] reinforce this limitation by stating that there are not many experimental works on the subject, and that the lack of coordination and group awareness (i.e., clarity about who work on what) intensifies the difficulties of controlling distributed projects. Moreover, they highlight the need of more studies on CM applied to the DSD context.

Therefore, the goal of this work is to investigate and gather knowledge on researches related to version control in the context of DSD by performing a systematic mapping study. We focus on investigating whether distributed teams use automatic tools and which commercial and academic solutions are applied in this scenario, as well as their features.

A systematic mapping study aims to evaluate and interpret all available knowledge relevant to a particular research question or topic by using a rigorous, auditable, and reproducible method [11], [12]. Furthermore, it aims at synthesizing and divulging research results, identifying missing or incomplete parts of the research, and determining the need of a complete systematic review [11], [12]. The guidelines provided by [12] and [13] are followed in this work.

This systematic mapping aimed at answering one main research question and four secondary research questions:

- **RQ: How is Version Control performed in Distributed Software Development?**
 - *RQ1: Which are the tools used for supporting version control in DSD?*
 - *RQ2: Which are the challenges related to Version Control in DSD?*
 - *RQ3: Which strategies, techniques, models, and processes are used to support version control in DSD?*
 - *RQ4: How was the described research evaluated?*

Altogether, 13 challenges, 7 tools, and 5 approaches were collected from 29 studies published between 2002 and 2012. The main contribution is a set of challenges, tools, and other solutions proposed for Version Control in DSD Projects. The selected studies show that practices, models, and tools to support Version Control in DSD projects are still few in the literature. Although many studies show interesting results on collaborative development, most of them do not mention distributed team. Few studies actually focus on versioning in DSD. Only these studies were considered in this research. This reflects that the research on this topic is still in its early stages and requires maturation.

This work is organized as follows. Section II presents the research methodology, whose steps are defined in the research protocol for systematic mapping. Section III shows collected data with general information about the selected studies. Section IV describes the systematic mapping results. Section V presents concluding remarks, along with result analysis and main contributions.

II. RESEARCH METHODOLOGY

This section presents the research method of systematic mapping study. This study was conducted from June to December 2012 and was extended in May 2013.

A. Research Steps

In accordance to the recommendations of [8], [12], and [13], the research was conducted through the following steps:

- 1) *Planning the Review*
 - Identification of the need for a review
 - Specifying the research question(s)
 - Developing a review protocol
- 2) *Conducting the Review*
 - Identification of Studies
 - Selection of Studies
 - Data extraction
 - Data synthesis
- 3) *Reporting the Review*
 - Specifying dissemination mechanisms
 - Formatting the main report

B. Search Terms

The search terms are built in three steps: structuration of research questions in terms of PICOC [13] (Population, Intervention, Comparison, Outcome, and Context) in order to identify keywords, identification of synonyms for each of the keywords, and build the search string based on the combination of the key terms and their synonyms, using the OR and AND operators.

Comparison and Context are not relevant in this work, since this mapping aims at conceiving an overview of the subject through an exploratory study. The result of this process is presented in TABLE I.

TABLE I SEARCH TERMS

Population	("Global software development" OR "Global software engineering" OR "Global software teams" OR "Collaborative software development" OR "Collaborative software engineering" OR "Distributed work" OR "Distributed development" OR "Distributed teams" OR "Geographically distributed software" OR Offshore OR Offshoring OR "Dispersed teams" OR "Dispersed Locations" OR "Multi-site development")
Intervention	AND ("Configuration management" OR "Version control" OR Versioning OR "System integration" OR "Integrating the code")
Outcome	AND (Tool OR Software OR Program OR System OR Model OR Process OR Framework OR Method OR Technique OR Approach)

C. Search Source

Some criteria, were taken under consideration to select the search engine. The search engines adopted in this mapping fulfilled the following requirements:

- They are capable of using logical expressions or a similar mechanism.
- They allow full-length searches or searches only in specific fields of the works.
- They are available in the researcher's institution.
- They cover the research area of interest in this mapping: computer science.

According to these requirements, we used IEEEXplore and Scopus as search engines.

In addition to the use of search engines, we also performed snowballing [14], [15] in this mapping. Snowballing is an evidence-based software engineering technique for finding relevant works based on the studies references (backward snowballing) and on works that actually mention the selected studies (forward snowballing). Google Scholar¹ was adopted to support this process.

D. Exclusion Criteria

Studies were discarded according to the following exclusion criteria:

- [EC1] Studies that do not contain information on Version Control in Distributed Software Development;
- [EC2] Studies that are neither freely available for download;
- [EC3] Publications that describe and/or contain keynote speeches, tutorials and courses.

E. Study Selection Process

The selection process was developed in four steps:

- Step 1: Initially, searches were performed and a list containing all the papers found was saved. This process was completed with the aid of the Zotero tool².
- Step 2: The researcher conducted an analysis of papers' titles, abstracts and keywords, ruling out the ones that met the exclusion criteria, and, thus, completing the First Filtration.
- Step 3: The researcher conducted the reading of the introduction and conclusion of all papers that went through the First Filtration, leaving out the ones that met at least one of the exclusion criteria, and, thus, performing the Second Filtration.
- Step 4: Finally, all papers have been entirely read and the ones that went through the former filtrations, but met any of the exclusion criteria at this point, were disregarded, thus, completing the Third Filtration.

The selection process utilizing the snowballing method was conducted in a similar way, starting from Step 2. After identifying the studies, the reading of their titles, abstracts

and keywords from the references, related studies, and most recent works referencing these studies, was performed. Moreover, clearly irrelevant references were immediately discarded.

F. Data Extraction and Mapping

Zotero was used in the management of the studies found. Data extraction forms were implemented using text editor to record information about how the studies answered the research questions. The extracted information from all the articles was: title, publishing year, full reference, and source. Also, discarded publications were marked with 'EC#', where # is the number of the exclusion criteria. The selected ones were marked with 'OK'.

Both quantitative and qualitative analyses were performed over the collected data. The quantitative analysis consists of the quantity of publications returned in the searches and filters. The qualitative analyses were regarding research questions.

III. COLLECTED DATA

This section outlines general information about the selected studies, such as year of publication, the research method, and the country of origin..

A. Number and Source of Studies

Step 1 (Section II.E) retrieved 102 studies from the search engines listed in Section II.C, but, among these, 17 studies were retrieved from both search engines, summing up 85 individual studies. After performing the document selection procedure described in Step 2 (First Filtration) 44 relevant articles were selected. In the Second Filtration, described in the Step 3, 23 relevant articles were selected.

Finally, the Third Filtration selected 11 relevant articles. Among those excluded studies, 12 were excluded by [EC3], 61 by [EC1] and only 1 by [EC2]. The distribution of these articles among the search engines is shown in Figure 1.

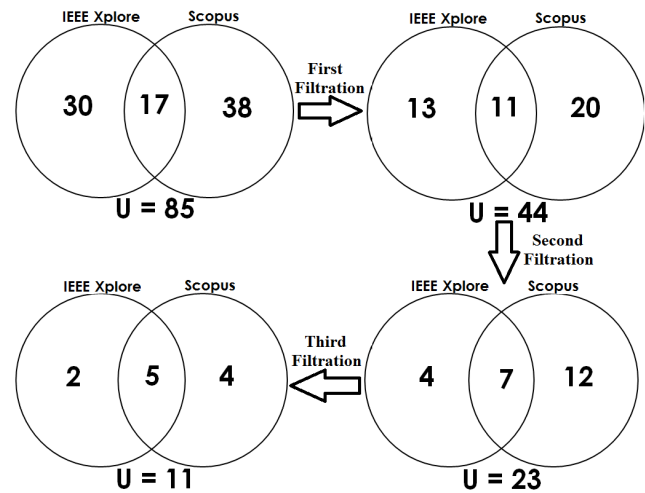


Figure 1 Number and Source of Studies

The backward snowballing method was applied to the 11 selected results. Forty-two studies were analyzed and 32 of

¹ <http://scholar.google.com/intl/en/scholar/about.html>

² <http://www.zotero.org/>

them passed the First Filtration. In the Second Filtration, with the reading of the studies' introductions, only 8 studies were selected. Finally, after the complete reading of the works, just one article met the [EC1] exclusion criterion and was ruled out. Therefore, 7 articles were selected by the backward snowballing method.

The 11 studies selected by the search in the research libraries listed in Section II.C were also used in the application of the forward snowballing method. In this process, 146 studies were retrieved. Among these results, 9 studies were considered potentially relevant, passing through the First Filtration. After that, 4 studies were regarded as relevant, thus, passing through the Second and third Filtrations.

After this, the backward snowballing method was applied again over the 11 new studies. One hundred and fifty-nine were analyzed and 19 of them passed the First Filtration. After the Second and third Filtrations, only one study was selected. In the forward snowballing method, 146 studies were retrieved and 24 were considered potentially relevant. After the Second and third Filtrations, only 6 studies were regarded as relevant.

The Figure 2 presents the results from the application of the snowballing methods. The technique showed itself effective, since it obtained in the first iteration the same number of articles as the search in the research libraries did: 11 results. In the second iteration, 7 new studies were obtained.

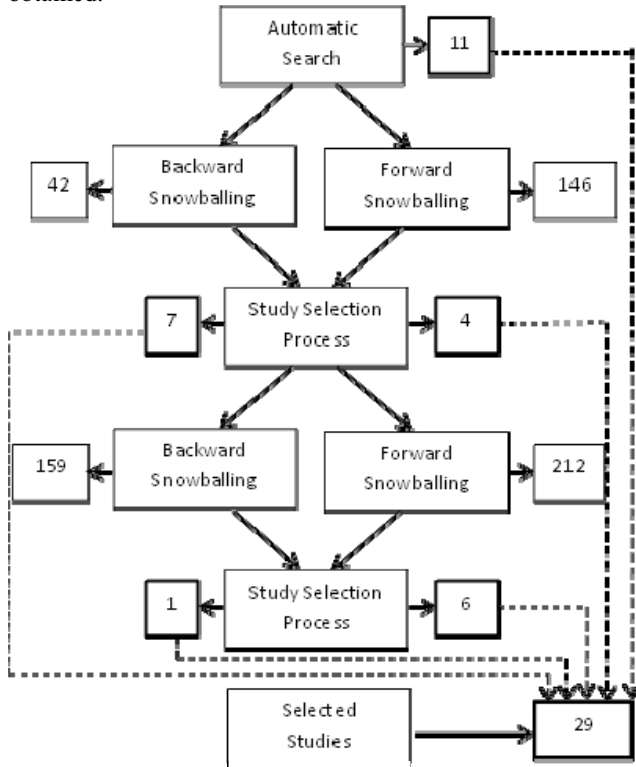


Figure 2 Snowballing methods

B. Temporal View of the Selected Publications

Although the search was not sorted by year, all the results selected were published within 2002 and 2012, which shows this topic has been gaining momentum lately. Besides, 25 of them (86%) were published after 2006, which coincides with the raising of new conferences on the theme, including the ICGSE.

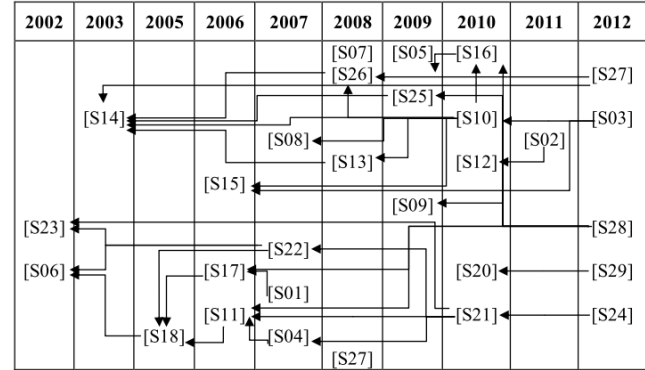


Figure 3 Distribution and Temporal relationship between the Studies

Figure 3 shows the selected studies' distribution over time. Moreover, the relationship among the selected studies can be observed. The papers [S10] and [S28], two mapping study, have most outgoing arrows: 6 and 5, respectively. The paper [S18], a study about Palantir tool, is referenced by five studies, including [S7], [S13], [S11] and [S22] from the same author.

C. Data Sources

Conference proceedings provided 21 studies (72%) and 7 studies (24%) came from journals. Noteworthy, one of the works included in this research is a Doctorate Degree dissertation. TABLE II shows the list of conferences and journals, that published papers included in the research.

TABLE II CONFERENCES AND JOURNALS

Type of Publication	List of Publishing Places	Amount of Studies	%
Conference	ICGSE (6), ICSE (3), SIGCSE (2), COMPSAC (2), APSEC, DiSD, ERCIM, FSE, ICCGI, ICPADS, SoSEA, STEW	21	72%
Journal	ACM Transaction on Software Engineering and Methodology (2); Software: Practice and Experience (2); Computing Research Repository; Empirical Software Engineering; Journal of Visual Languages & Computing	7	24%
Dissertation	PhD Thesis, University of California, Irvine, 2008	1	4%

Among those from conference proceedings, 6 are from ICGSE, 3 are from ICSE and the remaining 12 come from 10 different events. Seven papers retrieved from journals are from different sources.

D. Authors

The mapping counted 62 authors in the 29 selected studies. TABLE III presents the authors of studies about CM in DSD and the occurrences of each researcher as author of papers. This information's can help the identification of research groups interested in the subject.

TABLE III AUTHORS

#Papers	Authors
5	Andrea De Lucia, Anita Sarma, Fausto Fasano
4	André van der Hoek, Genoveffa Tortora
3	Rocco Oliveto
2	Bernd Bruegge, Christian Pendleton, Jan Magnusson, Lars Bendix, Timo Wolf
1	All the others.

E. Countries

The researches originated from 16 countries, as shown in the Figure 4. The greatest number of selected papers are from USA, Italy and Germany.

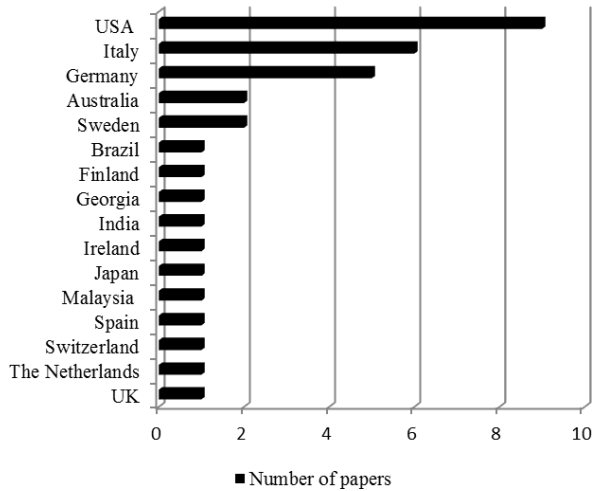


Figure 4 Countries of institutions of research's authors

IV. RESULTS

This section presents answers for each research question.

A. RQ1 – Which are the tools used for supporting version control in DSD?

This question aimed at identifying which tools are used as support to DSD projects. A search on general information of the tools was performed, including topology and type of collaboration.

TABLE IV ACADEMIC TOOLS SUPPORT

Tool	Description	Evidence (S1-S22)
Palantir	Palantir is a configuration management workspace awareness tool that provides developers with insight into other workspaces. Palantir itself is not a configuration management system and does not provide any traditional configuration management functionality such as artifact storage, workspace management, differencing and merging, or locking. Topology: Centralized Collaboration: Optimistic and Pessimistic	S6, S13, S14, S26, S27, S28
ADAMS	ADAMS (Advanced Artefact Management System) is an artefact-based process support system. ADAMS enables software engineers to create and store traceability links between artefacts. ADAMS supports the branching and merging of artifacts. Topology: Centralized Collaboration: Optimistic and Pessimistic	S11, S21, S22
SYSIPHUS	SYSIPHUS is a distributed environment providing a uniform framework for system models, collaboration artifacts, and organizational models. In SYSIPHUS, system models, collaboration artifacts, and organizational models are given equal emphasis and live in a single, shared repository. Topology: Centralized Collaboration: Not mentioned	S1, S17
STEVE and ADAMS	STEVE has been integrated in ADAMS to provide synchronous and asynchronous collaborative modeling functionalities. In particular, it allows developers to access and modify the same UML diagram at the same time, thus allowing distributed team members to discuss and model the system directly within ADAMS. Topology: Centralized Collaboration: Optimistic and Pessimistic	S4
CoDesign	CoDesign is a collaborative software modeling environment that supports system design in geographically distributed work settings. CoDesign's main contribution is an extensible conflict detection framework for collaborative modeling. Topology: Centralized Collaboration: Not mentioned	S12
Syde	Syde is a tool infrastructure to reestablish team awareness by sharing change and conflict information across developer's workspaces. Syde provides information of who is changing which parts of the system in real time - synchronous development. Syde is an extensible client-server application, where clients are Eclipse plug-ins that both capture changes and show change information as visual cues. Topology: Centralized Collaboration: Not mentioned	S16
CASI	CASI is a tool that informs developers about the changes that are taking place in a software project and the source code entities influenced by them. Topology: Centralized Collaboration: Not mentioned	S28

Seven academic proposals to support version control were identified. TABLE IV summarizes the tools. The first column shows the proposed tools for Version Control in DSD projects. The second column presents a brief description of the tools. The third one presents the selected studies that support the tool.

Palantir and ADAMS were the most cited tools. ADAMS was referred by 3 studies as an individual tool, and integrated with STEVE, a collaborative modeling interface, by one additional study. Palantir was also described by 5 complementary papers: a short paper in 2002, a tool in a full paper in 2003, 2008 and 2012, and a dissertation in 2008. Furthermore, it was listed in a mapping study in 2012. This tool's main purpose is to introduce awareness in existing configuration management system.

Another tool that appears in more than one publication was SYSIPHUS, whose main purpose is also to provide support to collaborative modeling. CoDesign and Syde are proposed to support Version Control in DSD projects. Additionally, CoDesign is focused on model versioning, Syde has its attention on source code, and CASI focus in indirect conflict.

All the tools were classified as centralized, that is, following the client-server model. ADAMS and Palantir were the only tools that present a description of the collaboration type they offer. A pessimistic approach to manage concurrency is adopted in ADAMS. However, it supports the branching and merging of artifacts.

Among the commonly adopted tools by the industry, 7 were referred to in the studies. Git, Jazz, Mercurial, Darcs, Perforce, Clearcase and Subversion (TABLE V) were described as version control tools focused on aiding DSD. Git, Jazz and Mercurial were utilized in three academic distributed projects, in which students were supposed to collaborate remotely.

TABLE V INDUSTRIAL TOOLS SUPPORT

Tool	Description	Evidence (S1-S22)
Git	Git is a distributed version control system, noted for its speed. Topology: Distributed Collaboration: Optimistic	S28, S29
Jazz Source Control	A platform for collaborative development created by IBM. Teams can choose to replicate changes to separate RTC Servers to allow for source code to be mastered in multiple locations for availability purposes. Topology: Centralized and Distributed Collaboration: Optimistic and Pessimistic	S5
Jazz and FriendFeed	The Jazz client extension with a Java wrapper of the FriendFeed API, a real-time feed aggregator that consolidates the updates from a number of social networking websites. Topology: Centralized Collaboration: Optimistic and Pessimistic	S25

Mercurial	Mercurial is a distributed version control system, noted for its well-balanced command set. Topology: Distributed Collaboration: Optimistic	S20
Darcs	Darcs is a distributed version control system. Topology: Distributed Collaboration: Optimistic	S28
Perforce	Teams at any location can transparently version their work as part of a collaborative workflow. Topology: Distributed Collaboration: Optimistic and Pessimistic	S28
Rational Clearcase	Clearcase is the market leader software configuration management solution that provides version control. ClearCase MultiSite enables file access across remote sites. Topology: Centralized and Distributed Collaboration: Optimistic and Pessimistic	S28
Subversion	Subversion is currently the most popular centralized open source version control system. Topology: Centralized Collaboration: Optimistic and Pessimistic	S28

B. RQ2 – Which are the challenges related to Version Control in DSD?

This question motivated the investigation of the challenges that DSD projects face when it comes to Version Control. Thirteen challenges on this matter were gathered. The challenges in the Version Control of DSD projects are summarized in TABLE VI. The first column shows the categories of challenges constructed from the data extracted from the evidences that are presented in the second column. The frequencies show the number of occurrences of each category. Each occurrence was given the same weight, thus, the frequencies merely reflect how many times a given category was identified in different studies, not how important it may be.

One important finding is that the first challenge listed was cited by half of the selected studies. Fourteen studies mentioned that dispersed software teams do not get information on what other teams are doing. For [S13], current CM systems promote workspaces that isolate developers from each other.

Another challenge, mentioned by nine studies, was the conflict detection delay. For [S16], only when a developer checks in his changes, will his colleagues have access to them and only when his colleagues synchronize their code with the repository, will they become aware of new changes.

Visualizing the traceability links between requirements was mentioned by 5 studies as a version control challenge in DSD projects. Software artifact traceability is the ability to describe and follow the life of an artifact (requirements, code, tests, models, reports, plans, etc.) developed during the software lifecycle [S22].

Working in different CM environments was a challenge cited by 3 studies. The decision to keep distinct CM environments for each team brought together several consequences [S15]. Communication delay was considered problematic in 2 studies. Projects with globally distributed members have to cope with communication delay due to physical distance to the server [S7].

TABLE VI CHALLENGES DETECTED

Challenge (C1-C13)	Evidence (S1-S22)
C1. Dispersed software teams do not get information about what other teams are doing (Frequency: 11)	S3, S6, S10, S17, S13, S14, S16, S18, S19, S21, S22, S25, S26, S27
C2. Non-real-time collaboration, since conflicts are only detected when the engineers “check-in” their changes (Frequency: 7)	S2, S6, S12, S16, S18, S21, S22, S24, S27
C3. It can be very difficult to visualize the traceability link between requirements. (Frequency: 5)	S1, S10, S18, S21, S22
C4. Working in different CM environments (Frequency: 3)	S3, S10, S15
C5. Dependency and Delay due to physical distance to the serve, mainly in centralized topology (Frequency: 2)	S3, S7
C6. Control over a distributed environment: Working distributed, there is always a risk that the development environment starts to diverge among the sites. (Frequency: 1)	S3
C7. When working distributed, there is a risk that different sites handles access control differently (Frequency: 1)	S3
C8. In centralized topology, changes on files get only backed up if another developer updates them into his repository. (Frequency: 1)	S7
C9. Not enough preparation time taken to set up CM infrastructure (Frequency: 1)	S8
C10. Provide the same information repeatedly to different tools: a developer makes changes to the source code and files a bug report in the issue tracking system. Additionally he might need to inform other developers about them. (Frequency: 1)	S9
C11. Artifacts with different versions and content at each site (Frequency: 1)	S10
C12. Many current version control systems are focused on textual (i.e., source code) documents (Frequency: 1)	S11
C13. Dependency between the developed modules (Frequency: 1)	S15

Some other important challenges were reported as well, but they were only mentioned in one study. Problems like not enough preparation time taken to set up the CM infrastructure, different versions of artifacts and each site, and dependency between the modules developed.

C. RQ3 – Which strategies, techniques, models and processes are used to support in version control in DSD?

This section presents the models and frameworks proposed in the literature to support Version Control in DSD projects. The first column shows the approach proposed to Version Control in DSD projects. The second column shows the approaches’ concepts. The third column presents the evidence on the selected studies.

TABLE VII PROPOSED APPROACHES

Approach	Description	Evidence (S1-S22)
RepoGuard	RepoGuard is a framework for integration of development tools with source code repositories. RepoGuard is written in the Python programming language, which allows for easy integration of other tools. The following version control systems are currently supported: Subversion, Git, and Perforce.	S9
Peer-to-peer version control	The decentralized peer-to-peer version control system is built on top of the p2p-framework FreePastry, which implements the Pastry overlay routing and maintenance.	S7
Conceptual Framework based on components	Framework for the evolution of software models in a collaborative modeling environment. It is built on BDI agent architecture framework which aims to maintain the consistency of project models and real-time conflict solving, given the changes made simultaneously by various devisor.	S2
OSCAR	OSCAR is an architecture for a distributed repository system to manage active artefacts. OSCAR defines active artefacts as two major components: the meta-data that describes their properties and the content (source code etc.). The four key modules within this architecture are: presentation, indexing, storage, metrics.	S23
Change Support Model	Change Support Model for distributed collaborative work is an approach that constructs an information repository to precisely reflect the state of work. It manages the states of artifacts/products made through collaborative work and the states of decisions made through communications. The information repository allows detection of inconsistencies and uncertainties.	S24

Two frameworks and approaches, and one architecture to Version Control in DSD projects were identified. A framework for integration of development tools with source

code repositories was proposed, as well as a framework based on agents to control changes. Furthermore, one decentralized peer-to-peer version control system was identified, an architecture and a model that supports changes in distributed repository.

D. RQ4 – How was the described research evaluated?

This question's purpose was to determine how the selected works were evaluated. Among 29 selected studies, most of them (9 studies) are university reports and were evaluated by undergraduate and MS students. Seven studies use examples, other 4 were not evaluated the proposal, and 3 articles are experience reports and experimental studies. Other 2 articles are systematic mapping. Only one study was evaluated combining both industry report and university report, as shown in Figure 5.

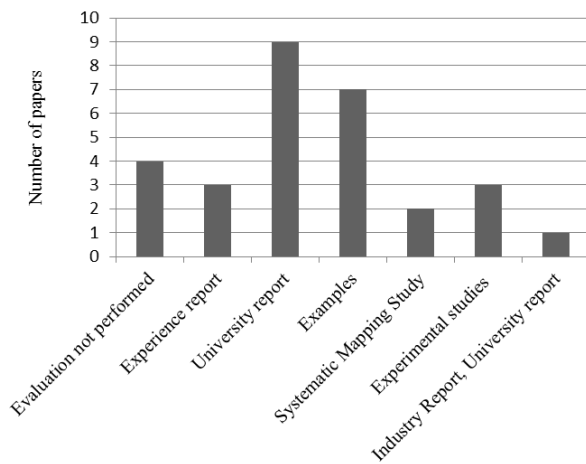


Figure 5 Research Evaluated

V. CONCLUSIONS

This section presents the final considerations and the discussions about the results. Furthermore discusses limitations of this research and further research.

A. Discussion about the Results

DSD is generally recognized as being much more challenging than traditional co-located development. Articles mentioned in this study ensure that the distributed environment heightens the challenges faced in traditional software development. In this context, CM has a critical role. The Version Control is used in all subsequent phases of software planning and developing. Development, testing, deployment, and installation are done based on the software configuration [1].

In the distributed environment, a greater effort is required to guarantee that all the people involved have a perception of the evolution of their work, and that the conflicts are resolved. Moreover, the people involved need to be sure that they are working on the last version of the project artifacts, and that there is no room for inconsistency in the project. It

is highly important that there is an aiding tool to version control that keeps the artifacts consistency and improves the work coordination.

An interesting reflection is made on [16], when the authors say that “CM was put into the world exactly to handle certain aspects of distribution on traditional projects”. They say that rarely requirement engineers, designers, testers, and programmers are sitting at the same place at the same time.

Nevertheless, there are only a few works on the topic that evidences what changes exactly in the Version Control for DSD projects, what the challenges related to the distributed environment are, and what are the proposed solutions to mitigate these challenges in this scenario.

This research's main question was “How is Version Control performed in Distributed Software Development”. It was asked in order to verify if the same tools used in traditional software development are also used in the distributed environment and whether they offer or not the needed support.

In this sense, this work's main contributions consists in a mapping of the challenges and solution proposals to support version control in the distributed development scenario:

#1: Academic tools to support Version Control in DSD projects

This research evidenced that researchers are concerned about allowing for a better perception of the work that is being developed by other remote team members, such as constant conflict verification when it comes to Version Control in DSD. Another spotted concern is present in the project design phase, especially the evolutions of UML models.

Seven academic tools have been identified (ADAMS, Palantir, SYSIPHUS, ADAMS+STEVE, CoDesign and Syde, CASI) and all of them are based in the client-server model. Whereas ADAMS, SYSIPHUS, ADAMS+STEVE, and CoDesign are mainly focused on collaborative modeling support, CASI, Palantir e Syde are focused on source code.

#2: Industrial Tools to support Version Control in DSD projects

Seven version control tools that are popular in conventional software development were indicated by two selected studies. The authors used the Git, Jazz and Mercurial tools in academic projects at which students worked in a distributed scenario.

Just a few studies identified in this research evinced the utilization of traditional co-located VCS. However, one cannot state that these tools are not used at all, but it is fact that their use is not frequently reported in the literature.

#3: Obstacles detected by the searches regarding Version Control in DSD

Thirteen challenges related to Version Control in DSD projects have been identified, some of which were mentioned more times than other by the selected studies, as dispersed software teams do not get information on what other teams are doing and the conflict detection delay.

The challenges that were mentioned several times are also the ones that are the targets of the proposed aiding tools. Palantír, for instance, aims at providing a wider perception of the work that is being developed in different workspaces.

#4: Few works evaluated in industrial environments and experimental studies

Most of the selected studies' proposals were tested solely in academic environments or merely represent simple use examples. This shows that is need a greater exchange between the academy and industry so that both can benefit from it. With this proximity, thus, academics can level up the maturity of their researches and propose more appropriate solutions to the needs of software companies.

B. Limitations

The main limitations and threats to the validity of this study lie in the fact that the mapping process was performed by only one researcher. This threat is considered acceptable by [14] for doctoral students that make use of this method. According to the text, it is sufficient that the thesis advisor engages in the protocol review and perform parts of the review himself.

In addition, from the 11 studies selected from the research libraries, the backward snowballing and forward snowballing were used only twice. The first application of the snowballing method resulted in 11 new studies. The second application of the snowballing method resulted in 7 new studies. Although continuing to apply this technique in these 7 new studies is expected from this research, it has not yet been possible due to time restrictions.

C. Further Research

The previously discussed limitations offer clear paths to further research. The utilization of the snowballing method in the 7 new studies can enrich the information collected so far on Version Control in DSD projects, with the empowerment of the evidence on the challenges and already listed solutions, as well as new proposals.

Besides, given the limited number of industrial reports, the conduction of a survey in software companies with distributed projects to identify how they perform the version control may display clearer conclusions on how Version Control is performed in Distributed Software Development.

ACKNOWLEDGMENT

The authors would like to thank CAPES, CNPq, and FAPERJ for the financial support.

REFERENCES

- [1] E. Carmel, *Global software teams: collaborating across borders and time zones*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1999.
- [2] L. Pilatti, J. L. N. Audy, and R. Prikładnicki, "Software configuration management over a global software development environment: lessons learned from a case study," in *Proceedings of the 2006 international workshop on Global software development for the practitioner*, New York, NY, USA, 2006, pp. 45–50..

- [3] S. ul Haq, "Issues in Global Software Development: A Critical Review," *Journal of Software Engineering and Applications*, vol. 04, no. 10, pp. 590–595, 2011.
- [4] M. Jiménez, M. Piattini, and A. Vizcaíno, "Challenges and Improvements in Distributed Software Development: A Systematic Review," *Advances in Software Engineering*, vol. 2009, pp. 1–14, 2009.
- [5] J. C. Binder, *Global Project Management: Communication, Collaboration and Management Across Borders*. Gower Publishing, Ltd., 2007.
- [6] B. Bruegge, A. D. Lucia, F. Fasano, and G. Tortora, "Supporting Distributed Software Development with fine-grained Artefact Management," in *Global Software Engineering, 2006. ICGSE '06. International Conference on*, 2006, pp. 213–222.
- [7] S. S. M. Fauzi, P. L. Bannerman, and M. Staples, "Software Configuration Management in Global Software Development: A Systematic Map," in *Software Engineering Conference (APSEC), 2010 17th Asia Pacific*, 2010, pp. 404–413.
- [8] F. Q. B. da Silva, C. Costa, A. C. C. França, and R. Prikładnicki, "Challenges and Solutions in Distributed Software Development Project Management: A Systematic Literature Review," in *2010 5th IEEE International Conference on Global Software Engineering (ICGSE)*, 2010, pp. 87–96.
- [9] J. young Bang, D. Popescu, G. Edwards, N. Medvidovic, N. Kulkarni, G. M. Rama, and S. Padmanabhuni, "CoDesign: a highly extensible collaborative software modeling framework," in *2010 ACM/IEEE 32nd International Conference on Software Engineering*, 2010, vol. 2, pp. 243–246.
- [10] B. O'sullivan, "Making sense of revision-control systems," *Communications of the ACM*, vol. 52, no. 9, pp. 56–62, 2009.
- [11] D. Budgen, M. Turner, P. Brereton, and B. Kitchenham, "Using mapping studies in software engineering," in *Proceedings of PPIG*, 2008, pp. 195–204.
- [12] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic mapping studies in software engineering," in *12th International Conference on Evaluation and Assessment in Software Engineering*, 2008, pp. 71–80.
- [13] B. A. Kitchenham and S. Charters, "Guidelines for performing Systematic Literature Reviews in Software Engineering," Keele University and University of Durham, EBSE Technical Report Version 2.3, 2007.
- [14] J. Webster and R. T. Watson, "Analyzing the Past to Prepare for the Future: Writting a Literature Review," 2002.
- [15] S. Jalali and C. Wohlin, "Systematic literature studies: database searches vs. backward snowballing," in *Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement*, New York, NY, USA, 2012, pp. 29–38.
- [16] L. Bendix, J. Magnusson, and C. Pendleton, "Configuration Management Support for Distributed Software Development," presented at the Proceedings of the Second International Software Technology Exchange Workshop, Kista, Sweden, 2012.

SELECTED STUDIES

- [S1] Berenbach and T. Wolf, "A unified requirements model; integrating features, use cases, requirements, requirements analysis and hazard analysis," in *Global Software Engineering, 2007. ICGSE 2007. Second IEEE International Conference on*, 2007, pp. 197–203.
- [S2] H. K. Dam and A. Ghose, "An agent-based framework for distributed collaborative model evolution," in *Proceedings of the 12th International Workshop on Principles of Software Evolution and the 7th annual ERCIM Workshop on Software Evolution*, New York, NY, USA, 2011, pp. 121–130.
- [S3] L. Bendix, J. Magnusson, and C. Pendleton, "Configuration Management Stories from the Distributed Software Development Trenches," in *2012 IEEE Seventh International Conference on Global Software Engineering (ICGSE)*, 2012, pp. 51–55.
- [S4] A. De Lucia, F. Fasano, G. Scanniello, and G. Tortora, "Enhancing collaborative synchronous UML modelling with fine-grained

- versioning of software artefacts,” *Journal of Visual Languages & Computing*, vol. 18, no. 5, pp. 492–503, Oct. 2007.
- [S5] A. Meneely and L. Williams, “On preparing students for distributed software development with a synchronous, collaborative development platform,” in *Proceedings of the 40th ACM technical symposium on Computer science education*, New York, NY, USA, 2009, pp. 529–533.
- [S6] A. Sarma and A. van der Hoek, “Palantir: coordinating distributed workspaces,” in *Computer Software and Applications Conference, 2002. COMPSAC 2002. Proceedings. 26th Annual International*, 2002, pp. 1093 – 1097.
- [S7] P. Mukherjee, C. Leng, W. W. Terpstra, and A. Schurr, “Peer-to-Peer Based Version Control,” in *Parallel and Distributed Systems, 2008. ICPADS '08. 14th IEEE International Conference on*, 2008, pp. 829 –834.
- [S8] R. Kommeren and P. Parviainen, “Philips experiences in global distributed software development,” *Empirical Software Engineering*, vol. 12, no. 6, pp. 647–660, 2007.
- [S9] M. Legenhausen, S. Pielicke, J. Ruhmkorf, H. Wendel, and A. Schreiber, “RepoGuard: A Framework for Integration of Development Tools with Source Code Repositories,” in *Global Software Engineering, 2009. ICGSE 2009. Fourth IEEE International Conference on*, 2009, pp. 328 –331.
- [S10] S. S. M. Fauzi, P. L. Bannerman, and M. Staples, “Software Configuration Management in Global Software Development: A Systematic Map,” in *Software Engineering Conference (APSEC), 2010 17th Asia Pacific*, 2010, pp. 404 –413..
- [S11] B. Bruegge, A. D. Lucia, F. Fasano, and G. Tortora, “Supporting Distributed Software Development with fine-grained Artefact Management,” in *Global Software Engineering, 2006. ICGSE '06. International Conference on*, 2006, pp. 213 –222.
- [S12] J. young Bang, D. Popescu, G. Edwards, N. Medvidovic, N. Kulkarni, G. M. Rama, and S. Padmanabhuni, “CoDesign: a highly extensible collaborative software modeling framework,” in *2010 ACM/IEEE 32nd International Conference on Software Engineering*, 2010, vol. 2, pp. 243 –246.
- [S13] A. Sarma, “Palantir: Enhancing Configuration Management Systems with Workspace Awareness to Detect and Resolve Emerging Conflicts,” UNIVERSITY OF CALIFORNIA, Irvine, CA, 2008.
- [S14] A. Sarma, Z. Noroozi, and A. van der Hoek, “Palantir: raising awareness among configuration management workspaces,” in *25th International Conference on Software Engineering*, 2003. *Proceedings*, 2003, pp. 444 – 454.
- [S15] L. Pilatti, J. L. N. Audy, and R. Prikladnicki, “Software configuration management over a global software development environment: lessons learned from a case study,” in *Proceedings of the 2006 international workshop on Global software development for the practitioner*, New York, NY, USA, 2006, pp. 45–50.
- [S16] L. Hattori and M. Lanza, “Syde: a tool for collaborative software development,” in *2010 ACM/IEEE 32nd International Conference on Software Engineering*, 2010, vol. 2, pp. 235 –238.
- [S17] B. Bruegge, A. H. Dutoit, and T. Wolf, “Sysiphus: Enabling informal collaboration in global software development,” in *Global Software Engineering, 2006. ICGSE'06. International Conference on*, 2006, pp. 139–148.
- [S18] A. De Lucia, F. Fasano, R. Francese, and R. Oliveto, “Traceability Management in ADAMS,” in *Proceedings of the International Workshop on Distributed Software Development*, 2005.
- [S19] L. Bendix, J. Magnusson, and C. Pendleton, “Configuration Management Support for Distributed Software Development,” presented at the *Proceedings of the Second International Software Technology Exchange Workshop*, Kista, Sweden, 2012.
- [S20] D. Rocco and W. Lloyd, “Distributed version control in the classroom,” in *Proceedings of the 42nd ACM technical symposium on Computer science education*, New York, NY, USA, 2011, pp. 637–642.
- [S21] A. De Lucia, F. Fasano, R. Oliveto, and G. Tortora, “Fine-grained management of software artefacts: the ADAMS system,” *Software: Practice and Experience*, vol. 40, no. 11, pp. 1007–1034, 2010.
- [S22] A. D. Lucia, F. Fasano, R. Oliveto, and G. Tortora, “Recovering traceability links in software artifact management systems using information retrieval methods,” *ACM Trans. Softw. Eng. Methodol.*, vol. 16, no. 4, Sep. 2007.
- [S23] C. Boldyreff, D. Nutter, and S. Rank, “Active artefact management for distributed software engineering,” in *Computer Software and Applications Conference, 2002. COMPSAC 2002. Proceedings. 26th Annual International*, 2002, pp. 1081–1086.
- [S24] P. T. T. Huyen and K. Ochimizu, “A Change Support Model for Distributed Collaborative Work,” *CoRR*, 2012.
- [S25] F. Calefato, D. Gendarmi, and F. Lanubile, “Embedding social networking information into jazz to foster group awareness within distributed teams,” in *Proceedings of the 2nd international workshop on Social software engineering and applications*, New York, NY, USA, 2009, pp. 23–28.
- [S26] A. Sarma, D. Redmiles, and A. van der Hoek, “Empirical evidence of the benefits of workspace awareness in software configuration management,” in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, New York, NY, USA, 2008, pp. 113–123.
- [S27] A. Sarma, D. F. Redmiles, and A. Van der Hoek, “Palantir: Early detection of development conflicts arising from parallel code changes,” *Software Engineering, IEEE Transactions on*, vol. 38, no. 4, pp. 889–908, 2012.
- [S28] J. Portillo-Rodríguez, A. Vizcaíno, M. Piattini, and S. Beecham, “Tools used in Global Software Engineering: A systematic mapping review,” *Information and Software Technology*, vol. 54, no. 7, pp. 663–685, Jul. 2012.
- [S29] X. Zhiguang, “Using Git to Manage Capstone Software Projects,” in *The Seventh International Multi-Conference on Computing in the Global Information Technology*, Venice, Italy, 2012, p. 159 to 164.