UNIVERSITY OF GOTHENBURG

Version Control Systems in Corporations: Centralized and Distributed

An explorative case study into the corporate use of version control systems

Bachelor of Science Thesis in the Software Engineering & Management Program

MY HÖGBLOM
VIKTOR GREEN

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
Göteborg, Sweden, June 2013

Version Control Systems in Corporations: Centralized and Distributed
An explorative case study into the corporate use of version control systems

My Högblom
Viktor Green

Examiner: Matthias Tichy
Supervisor: Henrik Sandklef

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden June 2013

Version Control Systems for Corporations: Centralized and Distributed

An explorative case study into the corporate use of version control systems

My Högblom
Gothenburg University, Sweden
my@student.gu.se

Viktor Green
Gothenburg University, Sweden
viktor@student.gu.se

ABSTRACT

Version control systems are important to all software development companies, and has been in use since the 1970s. This case study examines the effects version control systems have on two companies' configuration management strategies, and analyzes if a particular way of working influences the use of version control systems.

The study's main contribution is an analysis in how companies work with version control systems, and what drawbacks they see in their current systems. The analysis yields both expected and unexpected results, and the implications can probably be generalized to any software development company.

The units of analysis for this case study are two differently sized software development companies in Sweden. Between them they represent both co-located and distributed teams; and using both centralized and distributed version control systems.

Our conclusions are most notably that the type of version control system only has a limited effect on the configuration management strategy. We also found that features that are desired by companies are more user-friendly graphical user interfaces, atomic commits, better merging tools, integration with project lifecycle, and better handling of different file types.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management

General Terms

Configuration Management

Keywords

Version Control Systems, Distributed

1. INTRODUCTION

Version control systems (VCS) are often used in companies today for developers to store their source code, documentation, and other files. Currently there are basically two types, or philosophies, among VCSs: centralized systems (e.g. Perforce or Subversion) and distributed systems (e.g. BitKeeper or Mercurial). The fundamental difference is that a centralized VCS has a central repository that all developers connect to, while in a distributed VCS each developer has their own complete copy of the repository.

As companies are always looking to improve their productivity they are increasingly looking into changing their VCSs to newer distributed systems. Companies therefore need an unbiased analysis of different solutions in order to select a VCS based on research rather than personal preference.

Currently there is very limited research done in this area, and most articles focus on the open source perspective. Researchers therefore need to study what advantages and disadvantages different tools and philosophies have for different ways of working.

For this study we are interested in finding out how companies choose between centralized and distributed VCSs, as well as features that they feel are lacking in current systems. For example, will a company with a co-located development team use a centralized VCS? In addition, will a company with teams distributed across the globe prefer a distributed VCS? Another perspective that we are interested in is how the configuration management (CM) strategy at a company affects the use of VCSs and vice versa.

This leads us to the research questions for this study:

- *To what extent does the type of VCS affect the CM strategy at a software development company?*
- *What features are desirable of a future VCS for companies?*

These research question are interesting to study in order to see the effects VCSs have on different types of development teams. Another interesting aspect is whether or not the advantages and disadvantages of distributed VCSs in comparison to centralized VCSs, affect different CM strategies in the same way.

The rest of this study is outlined as follows. First a background and review of related work is provided. Next the method for data collection and analysis which we followed is described. In the next section the results of the data collection and analysis are displayed. Subsequently, the discussion of the results are presented, and finally, the conclusions which were drawn from our results are listed.

2. BACKGROUND

In this section we provide a brief history of VCS relevant to the study, as well as a theoretical background to our study.

2.1 History of VCS

Version control began as local source code management on a mainframe computer with connected terminals, e.g. Source Code Control System (SCCS) [1, 21]; this type of version control kept all metadata in the actual source code and used locking to inhibit concurrent changes [21].

In the nineties, as personal computers became popular, VCSs evolved into systems that would check in and out against a centralized server, and Concurrent Versions System (CVS) was the first implementation of a centralized VCS as we see them today [1, 13]. These centralized systems brought possibility to access the system from a remote computer, and started out as nothing more than scripts that wrapped the function of a local VCS [13].

In the mid 1990s the proprietary centralized VCS, Perforce saw the light of day. It has since become a well known VCS, used for some time by both Microsoft and the Perl [1] project [8, 16, 24].

Due to the lack of a better free alternative, CVS had in 2000 become the de facto standard in the open source community [6]. At this time, a few developers set out to create Subversion with the explicit goals of trying to fix or replace CVS [1, 5, 6]. In the mid 2000s, Subversion had already gotten a considerable user-base, and surpassed CVS as the first choice of VCS [5, 16].

A second paradigm shift took place in 2002 when BitKeeper was released as the first distributed VCS [1]. Developers now had their own working copy of the repository locally, and this was then synced and merged via the VCS. BitKeeper was adopted early by the Linux project, but in 2005 Linus Torvalds decided to create a new VCS when BitKeeper became what he called *politicized* [15]. What he meant by this was that *"the relationship between the community that developed the Linux kernel and* [BitMover[2]] *broke down, and the tool's free-of-charge status was revoked."* [4].

The new VCS Torvalds created was Git; a distributed VCS that lately has increased in popularity [4]. In a survey from 2011, Git was the third most used VCS, after CVS in second and Subversion in first place (with more than 50% of the users) [10].

[1]Perl is an open source general purpose scripting language, initially developed for use on Unix.

[2]BitMover is the commercial company that develops BitKeeper

At the same time as Git, another distributed VCS was born: Mercurial [19]. Mercurial, however, has not reached the same popularity as Git; in the 2011 survey, Mercurial is one place below Git (a place Mercurial shares with Perforce) [10].

In the same survey taken 2012, Git saw even bigger numbers than the year before. Git was at this time the second most used VCS, with more than a fourth of all respondents saying they are using Git as their primary VCS [11]. This indicates that distributed VCSs are quickly gaining popularity. On the other hand Mercurial stayed in fourth place, but this time with a lower score [11]. This might indicate that it is not distributed VCSs in general that are gaining popularity, but rather that the specific VCS Git is gaining popularity.

2.2 Centralized vs. Distributed

Both centralized and distributed VCSs are largely in use today, although centralized version control is the most common [8]. Centralized systems work based on the client/server model and lets users work with code from a single central repository, while using either locking or merging to handle synchronization [14]. Rodríguez-Bustos & Aponte, and de Alwis & Sillito identified challenges surrounding the use of centralized VCSs [8, 22]:

- They require a network connection to work on the source code
- Developers must ask for permission in order to contribute to a project
- A single point of failure is an issue when using one server

Distributed systems on the other hand, let the user download a copy of the repository locally. Changes can then be merged and pushed into a master repository [1, 8, 14]. The biggest benefits of distributed VCSs is the possibility to work offline, and that they allow non-core committers to contribute to the development [8, 22].

We have not found any research articles that identify the disadvantages of distributed VCSs. However, on the online forum Stack Overflow, we found that the two most common complaints about distributed VCSs are that:

- Pessimistic locks are not available [25].
- They have weak tools for binary files [20].

The distributed VCSs Git and Mercurial were developed as a reaction to that BitKeeper was no longer free to use. These tools have been adapted to open source projects[3] to fit their specific needs. Some of the most important factors for open source projects are being able to work offline, using changesets instead of full versions, and flexibility in using different workflow models.[22]

[3]For example Mozilla, Python, KDE, NetBeans, Eclipse, GNOME, which used to employ CVS or SVN

3. METHODOLOGY

This section provides the details of the research design for this case study. We chose to conduct a case study in order to investigate the reasons behind choosing VCSs, and also to gather understanding about how different VCS types affect CM strategies [9, 23]. This study is based on qualitative data collection composing of interviews, and the data was analyzed using thematic analysis. To underpin the study we used a social constructivist ontology [7].

To the best of our knowledge, this is the first study done on this topic, which is the use of VCS in corporations. Due to the lack of research done in this area, a systematic literature review was not possible to conduct. Instead we described the available knowledge from this field in the theoretical background. We also chose to do an exploratory case study since this can be considered an initial investigation into this area of research. A case study offers in-depth understanding of a certain phenomena within a real life context [9, 23].

3.1 Research Setting

In this case study we interviewed employees that work with VCS on a daily basis at two software development companies that are based in Sweden.

The smaller company (henceforth called company A) has co-located development teams and a total of about 20 developers. They have used a variety of VCSs, and in 2009 decided to switch from the centralized VCS Perforce to the distributed VCS Mercurial. The switch was completed in early 2011, when all new development was using Mercurial. Currently, three different VCSs (Mercurial, Perforce, and Subversion) are being used for different tasks.

Company B is a larger global company that has both distributed and co-located teams of up to 40 developers, and a total of about 200 within the company. In this case, global refers to the fact that they have *"multiple development teams, spread over different time zones"*. We interviewed employees from two different large projects at the company, each having their own CM. At company B Subversion is most widely used, whereas other centralized systems such as CVS occur at certain sites. They are currently doing a pre-study regarding the possibility of migrating to Git for future projects.

The relative sizes of these companies affect the flexibility that they have in changing VCS, and also the time it takes to implement such a change. This will also affect the requirements of the VCS, as well as the CM strategy which they choose to adapt.

3.2 Research Questions

Our aim in this study is to answer the following research questions:

RQ1: *To what extent does the type of VCS affect the CM strategy at a software development company?* The first question is the basis for this study, and captures the essential connection between VCSs and ways of working.

RQ2: *What features are desirable of a future VCS for companies?* The second research question tackles the issue from another side, and tries to figure out what CM strategies require from the VCS.

We have two propositions that are based on our research questions. They state what the study is intended to show, and guides the selection of cases and the types of data to collect [9].

Proposition 1 (Related to RQ1): *If a company works in a centralized/distributed way, the company will find that a centralized or a distributed VCS respectively will work better for them.* We expect that the correlation between the process and VCS type to be strong. Research shows that open source projects, which are distributed in their nature, are eager adopters of the distributed VCSs [8, 15, 22]. This fact is congruent with this proposition, and thus we assume that we will find the same relationship in the corporate world.

Proposition 2 (Related to RQ2): *Companies have different feature requirements than the open source world.* Since companies often work in a different way than open source projects we assume that they also have different requirements from their tools.

3.3 Case Selection and Units of Analysis

In selecting the case for this study, we considered how the companies were structured, since we believed this would affect the choice of VCS. Also, the size of the company in terms of the amount of developers and other employees working with the VCS was considered.

We chose two companies which were on the opposite sides of the spectrum regarding both the factor of size and way of working (co-located versus distributed). This is the contrast that sparked our interest in this study, and makes comparison interesting.

Our case is the choice and usage of VCSs in the corporate world. The units of observation in our study are on the individual level with the employees at two companies [26]. However the units of analysis that our case study is based on are on a higher level; specifically, the two previously mentioned companies [23, 26]. This is in order to gain more insights into the choices and thinking at the company level.

3.4 Data Collection

The following section will describe how the collection of data for this study will be conducted.

3.4.1 Pre-Study

The first phase of the research delves into the current knowledge and research within this area of study. We searched IEEE Xplore, Springer Link and ACM Digital Library journal databases for research articles. Initially we searched using the search strings "version control system", "distributed version control system", and "centralized vs. distributed version control system" [4], as we felt that they were suitable to our study. When we realized that there was not much research done into the corporate use of VCS, we widened the scope. We decided to find studies related to the VCSs used

[4]And other variations of those, such as VCS, DVCS, distributed VCS, versioning, etc.

by the two companies in this study. The following keywords were used to search at this stage: mercurial, git, subversion, and perforce. We excluded articles by reading their abstract and found a few articles relevant to our research questions.

3.4.2 Interviews

The second phase of the research included a set of semi-structured interviews [18]. This was the main source of data collection for this study. The interviewees were selected to encompass the VCS from different perspectives: from the configuration manager's point of view who has a more administrative role, as well as developers and testers, who are users of the system.[23] We intended that each role should be represented by at least two persons if possible. And only interviewees with knowledge and experience of VCS were chosen.

The interviewees are categorized into roles; Table 1 shows the distribution of the roles for our interviews.

Both face-to-face interviews and email interviews were conducted in order to get as big a sample as possible, while still retaining the quality and depth in responses. The interviews were scheduled to take maximum one hour, and were held by two interviewers. Each interview was transcribed verbatim from the audio recording, and the script was used when analyzing the interviews. The interviews were carried out at the workplace of the interviewees, and were recorded unless the interviewees had objections. The interview questions were iteratively improved as each interview passed, in order to keep the study in line with the research question. For the email interviews, the interviewees were given one week to respond to the questions we sent them. Followup questions were sent by email to any interviewee when needed. All interviews were conducted in Swedish, and therefore all quotes have been translated to English by the authors. The interview questions are available in Appendix 1.

At company A, a developer was interviewed face to face, and the others got interview questions sent by email. A second developer at company A got email questions but did not answer them. Due to lack of time, we were not able to find a replacement for this interview; and as such the ratio of interviewees are skewed towards company B. We held two face-to-face interviews at company B, one with a developer and the second with both a developer and a CM.

3.5 Data Analysis

The following section will go into detail about how we analyzed the data in this study.

3.5.1 Thematic Analysis

Braun and Clarke [2] advocate thematic analysis in qualitative research, such as case studies like this. Thematic analysis is a good fit for exploratory studies because the findings are grouped in themes, which are easily understood. Grouping the data by theme gives a good overview of the qualitative data, as well as visualizing the data. This study followed the steps of thematic analysis, as described by Braun and Clarke [2]. The following is a description of how we implemented each of the steps.

Familiarizing yourself with the data.
This first step included the transcription of interviews as well as reading and re-reading the data. Towards the end of this practice we started writing down ideas for the coding to be done in the next step.

Generating initial codes.
At this point we went through our data again, in order to find *codes* [2]. What we realized during this process was that the ideas we wrote down earlier already seemed to be the codes we were looking for at this point.

Searching for themes.
In the beginning of this step we had a long list of codes that we categorized into different *candidate themes* and *sub-themes* [2]. After this we created an initial map of the connections between the themes. During the process of drawing the thematic map, we discovered that a set of super-themes could be abstracted. All our candidate themes were connected through the super-themes.

Reviewing themes.
When reviewing the themes we had, we realised that our candidate themes were actually sub-themes, and that the super-themes really were our candidate themes. What had previously been sub-themes were removed or included into a higher level theme. During this step we went through some iterations of our thematic-map, and a simplified version of that map can be seen in Figure 1.

Defining and naming themes.
In this step we iterated over the names of our themes, and started writing the *story* [2] of each theme. At this point some sub-themes were removed, as we realised that we did not have enough data to support them.

Producing the report.
This final step started when we had the basic descriptions (or stories) of all our themes and sub-themes. The output of this step is the Results section in this study.

Some of these steps are inherently iterative in their process, and this was utilized by us even more in order to start analyzing data before we had received all the interview responses.

3.6 Validity

The following section describes different threats to the validity of this study. We also include the measures which we have taken to avoid these threats.

3.6.1 Thematic Analysis

During the analysis we were aware of the most common pitfalls while conducting a thematic analysis, as they are described by Braun and Clarke [2]. This awareness allowed us to circumvent potential validity issues of thematic analysis. We also follow the criteria for a good thematic analysis, and use an iterative process to avoid erroneous conclusions and improve the validity of this case study [2].

3.6.2 Pre-Study

There is not a large amount of research written in this field, and most of the articles compare specific tools within the

Table 1: Roles of interviewees

Role	Number of interviewees		Description
	Company A	Company B	
Configuration Manager	1	2	This role is concerned with creating a strategy for working with VCS, as well as managing the branching strategy.
Developer	1	2	The developer interacts with the VCS daily as a user.
Test Manager	1	0	The test manager also interacts with the VCS on a daily basis, similar to the developer. But with more focus on building certain versions of the software for testing.

same type of VCS. This will have an effect on the background of this study in that it will not be as complete as it otherwise could have been; on the other hand since this is more of an exploratory study, and the purpose is to provide new knowledge where it was previously missing, therefore it is not as big of a threat to the validity as it would seem.

3.6.3 *Interview*

A potential threat to the validity in this study is that interviewees are selected through convenience sampling, rather than using a true random sample, which could skew the result [7]. However, Easterbrook states that *"Case study research uses purposive sampling rather than random sampling."* [9] This means that the relevance of this threat is diminished.

On the other hand, Easterbrook presents another threat to the validity of case studies: *"The major weakness of case studies is that the data collection and analysis is more open to interpretation and researcher bias."* [9] An effort to mitigate this risk was taken by selecting interviewees from two companies using different styles of version control systems. And as mentioned in the section about the validity of our analysis, we did multiple iterations over the data. That helps the validity of the analysis, but not the bias. The bias is difficult, or almost impossible, to avoid in qualitative studies. We are aware of this problem and therefore attempt to be objective.

Another threat which is presented by Runeson & Höst is the external validity, which concerns to what extent it is possible to generalize the findings.[23] The amount of interviews as well as the choice of case and units of analysis affects this threat. As mentioned before, we chose two quite different companies, as well as different roles within the company in order to investigate differences and different viewpoints. A greater number of interviews always increases the accuracy of the data. We do not feel that our study is large enough to be able to draw very generalized conclusions, but we do feel that they can be applied to the two companies in our study.

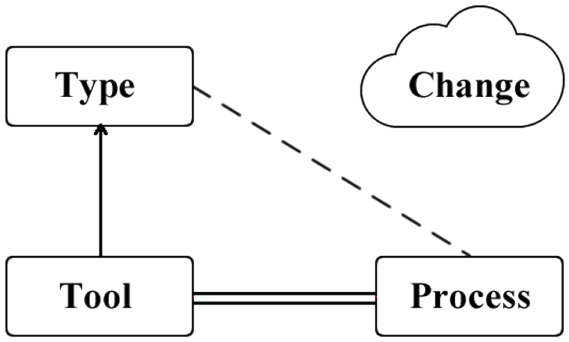
Yet another threat to the validity of this study is the fact that we may not get as many interviews as we plan for. As we are asking employees to get interviewed, some may not have the possibility to take time off from work during the timeframe we have available. To handle this threat, we have several email interviews, which lets the interviewee answer the questions at his or her own discretion.

4. RESULTS

This section presents the results attained from the thematic analysis on the data collected through interviews with seven people at two different companies. During the analysis we identified four main themes, each further divided in a set of sub-themes.

The themes can be seen in Figure 1 and are: *Type*, *Process*, *Tool*, and *Change*. Type has a parent child relation to Tool, or rather, the connection between the two is that each tool is of a certain type. Both Type and Tool are connected to Process; the Type–Process connection is a surprisingly weak connection even though we expected otherwise, and the Tool–Process connection is the most dominant connection. Change, which is the last theme, is not connected to any of the other three themes. It is displayed as a cloud because it is not the main focus of our study, and is not directly related to Type, Tool, and Process. Table 2 lists all the sub-themes that we have identified, and what main themes they are associated with.

Figure 1: Thematic map.



4.1 Type

This parent theme was identified because some of the codes that we tagged our data with could be grouped into this more general theme. The theme represents data regarding the different philosophies, or *types*, of VCSs. As described in the background there are basically two different types of VCSs: centralized and distributed. This section will also report our findings regarding the connection between the type and the process.

Table 2: Sub-Themes

Theme	Sub-Theme
Change	Process Tool Type
Process	Centralized Way of Working Smoothness
Tool	Branching Cost File Types Functionality GUI Hybrid Integration Performance
Type	Centralized Distributed

4.1.1 Centralized

Company B uses Subversion (a centralized VCS) and even though they have developer teams across the globe they feel that it works well. One of the developers expressed his feelings towards centralized systems as: *"Currently it works okay, with what we have."* A CM went into a bit more detail in his reasoning: *"As long as the geographical distance isn't too large and the network connections can handle high enough capacity, then I don't see any obstacles with centralized version control systems."* The TM from company A was also positive about using centralized VCS and expressed a belief that it is easier for a smaller company to use it since everyone works against a single repository that is always up to date.

4.1.2 Distributed

Distributed VCSs are mainly newer than their centralized counterparts. They also often have better tools, with more advanced features. One example, mentioned by many of the interviewees, is that distributed VCSs are superior when it comes to branching and merging. This was in fact the main reason for company A to switch from Perforce to Mercurial in the first place.

Merging is an intrinsic part of a distributed VCSs since each user has a copy of the repository, and in order to share the changes the users has to merge their repositories together on a regular basis. This can be done in different ways; a centralized VCS only allows its users to merge to the central server, while a distributed additionally gives the possibility to merge directly between the users. A developer at company B had the following to say about this: *"I do not see any direct disadvantages with distributed VCSs since they also allows for a centralized way of working."* Company A follows this principle. One developer described how they work with Mercurial: *"We are not using the power of the distributed, in my opinion. We are using it, in my eyes as a developer, in the same way* [as a centralized VCS]*; we check out from the central repository, and check it back in. It is very very seldom we are pushing between developers."*

Another aspect of distributed VCSs is that they can be harder to embrace, which can be an important factor when changing VCS. Two specific reasons were mentioned during our interviews. One reason was that distributed VCSs represent a different mentality from a centralized VCS. The second reason was that *"Distributed version control systems have a somewhat higher learning curve, which can present an obstacle in switching"*, as described by a CM at company B. This reason was also mentioned by an interviewee at company A.

Even though these systems may be more difficult to use, a couple of interviewees expressed their confidence in the endurance of distributed VCSs. One opinion was that distributed VCSs are here to stay since they give developers the possibility of working from out of the office. Another opinion was that these systems will grow even more in the future due to high involvement of the open source community.

4.1.3 Type and process

To our surprise we did not find a strong connection between the type of VCS and the CM strategy. The only instances were from company A, where the TM said that the type does not affect the way of working in any noticeable way, although specific tools can make it go more smoothly. The CM at company A said this about the suitability of the VCS to their way of working: *"The general answer is no, the distributed system we are using is not suitable. The reason for moving to the distributed system was taken when the CM strategy was weak, and we only looked at individual feature sets in the tools that did not necessarily have to do with the type of VCS."*

4.2 Process

Just as the theme *Type*, this theme is an abstracted theme. In this case, it is intended to encapsulate the different themes that we found regarding how the companies are working. In terms of VCSs the process is exemplified in CM strategies. There are multiple aspects to a CM strategy, and some of them will be discussed here. In mirroring the type of VCS, we see CM strategies as divided between centralized and distributed ways of working. This section will also include the findings regarding the connection between tools and the process.

4.2.1 Ways of Working

By a centralized strategy we mean that all the developers are co-located, and a distributed strategy would therefore imply a company having developer teams distributed across multiple locations, or team-members from the same team being distributed. Company A is rather small, has all developers co-located in the same office; as the developer eloquently said: *"We use a distributed system, but we work in a centralized way."*

The TM at company A expressed the opinion that it is easier for smaller companies to work in a centralized way: *"In my opinion, it is easier for a smaller company to use centralized version control systems since everyone always works against a 'canonical' midpoint and everything that gets checked in is always up to date."*

One of the CMs at company B said that *"Our organization is – like many other organizations – global in the sense that there are a multitude of developer teams spread out over different time zones. A* [centralized] *VCS is in that context clearly inferior to a distributed VCS".*

4.2.2 Tools vs. Process

A CM at company B said that they have strategies for VCS processes, but not for choosing specific tools. At company A, the TM expressed concern about using multiple tools: *"It is a big problem that we currently use a large number of different VCSs, which makes it difficult to get an overview. The reason for this is that some VCSs are better than others in processing certain types of data."*

Both companies appear to be happy with their processes and did not express any desire to change their way of working. However, CMs and developers from both companies did point out that new tools are a different story altogether. If there would be any system that would make their daily work easier, or solve drawbacks in their current systems, they would be more than willing to adopt something new. This implies that the process overrides the tools used.

Be that as it may, a CM at company B explicitly said that the VCS affects the CM strategy, and that the strategy they use is based on the fact that they use Subversion. A developer at company B backs up this story: *"We have adapted our strategy based on the tool we use."* The CM at company A expressed a similar opinion: *"We must adapt the CM strategies we have, based on the tools available from each VCS distributor."* And on a slightly different, but connected note, the developer at company A said: *"I'm not even sure how a different distributed VCS would change our way of working."*

However, one CM at company B was consistent and adamant about that the process comes first, and tools second. The following quote demonstrates this: *"I have no preference regarding VCS. The most important thing, in my opinion, is to have an explicit strategy to facilitate the business requirement. [...] Based on the requirements, a strategy is chosen and thereafter tools that are best suited to the strategy are selected."* In the same line of reasoning, the TM at company A expressed the opinion that if they would change VCS the new one would be a centralized VCS, since they work centralized anyway.

4.3 Tools

We found this theme to be the most important theme in the analysis, or at least the theme with most data points. This was a bit surprising since it is not very related to our main research question. We did not find any connection between the tools and the type in the data, but we maintain that there is such a connection, if only in the sense that each specific tool is of a certain type. This section will describe our findings with regards to desirable features in VCS tools, and also some reasoning for selecting one.

To further subdivide this section for the reader's peace of mind, we have grouped our sub-themes into three larger subsections: functional requirements, non-functional requirements, and future. The sub-themes that are part of functional requirements are: branching, file types, and functionality. Non-functional requirements include cost and performance. While future focuses on what the interviewees saw in a future VCS system: GUI, hybrid, and integration.

The CM at company A gave us a good description of their criteria for selecting a VCS. Although, how the different aspects are weighed against each other were left out of the description. The following list shows the areas company A uses when evaluating a VCS.

- *For the developer: How effectively are the daily operations performed? (Update, Merge/Resolve, and Commit)*
- *For the CM: Administration of branches and users. Visualizing changes on source code level, i.e. overviewing the history on all levels between branches and files.*
- *For IT: Licencing costs and the resources required by the VCS.*

4.3.1 Functional Requirements

One developer at company B thinks that Git is better than Subversion and CVS when it comes to merging, but that the biggest difference is the tools. When the same developer compares Subversion and CVS, *"the biggest difference is that I used CVS in the beginning of the 2000s and Subversion in the end, and that the tools around them got better with time. Therefore, I tend to think that Subversion is better, but honestly I'm not really sure."*

As mentioned in the above section on process, branching is an important part of a VCS. It is so important that, as mentioned previously in the section on type, branching was the main reason for why company A changed from Perforce to Mercurial. A developer at company B stated that branching in general is not a big problem in Subversion, although the developer has not tried any other types of VCSs. Another developer at company B had a slightly different opinion: *"It is said that branching works better in Subversion* [than in CVS] *, but I find the processes of branching/merging to be heart-stopping exercises in both tools."*

When company A migrated away from Perforce to get better branching capabilities, they noticed that Mercurial did not handle graphics files very well. Mercurial sees them as binary objects and needs to keep the whole file for each version, unlike Perforce which can recognize layer changes and other file type specific changes and keeps changes to these files as change deltas. Because of Perforce's widespread use in the gaming industry, it has adapted features specifically desired by this industry, like graphics file handling. The developer at company A expressed the belief that it is common knowledge that distributed VCSs do not handle binary files very well, since each user has a complete history of the repository on his or her computer. Without change deltas for binary files, the size of the repository will inevitably grow very fast. The TM at company A said that it is easier to use a centralized VCS, due to their use of graphics files.

Another aspect is the merge tool, an often used tool when interacting with all types of VCSs. *"In Mercurial you can select the merge tool that you want to use, and because of*

that, the majority here still use the Perforce tool, since the built in merge tool in Mercurial is pretty bad." (Developer at company A)

4.3.2 Non-Functional Requirements

When company A realized that Mercurial was not the optimal solution for their need, they split up their data into graphics and code. In order to save costs, now that Mercurial had replaced Perforce as their main VCS, they decided to move their graphics into Subversion. As the CM at company A put it: *"A drawback* [of Perforce] *is the cost per user"*. Another thing that became apparent after switching from Perforce was that Mercurial had much lower performance for operations like *check-out*. Both cost and performance was also mentioned by the CMs at company B. Cost being one of the main reasons for selecting non-proprietary VCSs, and performance being somewhat troublesome when teams situated in different locations were communicating with the server in Sweden: *"*[Centralized] *VCSs requires constant communication between the workstation and the server for these operations* [merge, diff and history]*, which leads to a slower way of working."* These negative performance aspects of working with a centralized VCS, can be abated with the use of distributed VCSs where most operations are performed on the local repository.

The CM at company A brought up an interesting point, and said that administration tools are better in centralized VCSs and that *"administration tools are really important, but often forgotten while selecting a VCS"*. This also led him to believe that *"centralized systems are better at scaling for big companies, from a CM point of view"*. However, this point was somewhat contradicted by one of the developers at company B: *"Subversion and CVS scale badly, due to their centralized nature."*

4.3.3 Future

To follow the line of thought that different VCSs are good at different things, the TM at company A suggested that in the future, an umbrella system for VCSs would be useful. It would give the user a unified view of everything under version control, but use different VCSs in the backend to utilize the advantages of each tool. A similar thought was given by the developer at company A, who said that the future of VCSs would contain some kind of hybrid between centralized and distributed VCSs. One example given was to have local commits, and a local history to use when there was no network connection available, but still keeping the better handling of binary files in centralized VCS. This feature was also something that a developer at company B felt was missing.

Similar to the ideas of umbrella and hybrid systems, is the notion of integration and plugins that was brought forth by interviewees at both companies. One such example would be the CM at company A who wished for better integration with bug tracking systems and project management tools.

Other features that the interviewees felt are lacking in current implementations of VCSs are *atomic commits* and *locking functionality for distributed VCSs*. Atomic commits are already standard in most VCSs [12]. Locking in distributed VCSs on the other hand is something not present in current systems. The reasoning for this wish, according to the developer at company A, is that before changes can be *pushed* to a non-local repository any previous changes must be *pulled* and *merged*. In this time span, another developer could *push* his or her changes to the same repository, forcing the first developer to repeat the process with the new changes now available. For a centralized way of working, a locking mechanism that could prohibit other users to *push* changes would instead allow each developer to complete his or her *pull*, *merge*, and *push* process without interruption. [3]

When talking about the future of VCSs, a large majority of the interviewees from both companies mentioned visual enhancements and graphical user interfaces (GUIs) as very desirable. A CM at company B stated that improved GUIs should focus on facilitating work for both developers and CMs. The motivation for the CM is to get better overview and history, and developers would gain by getting more easily comprehensible merging and branching views. The TM at company A said that *"Perforce is one of the better VCSs I have used, mainly due to its user-friendly UI"*.

4.4 Change

Change appeared in our analysis somewhat against our will. We were not focusing on change processes regarding VCSs, but the more we iterated over the data the more it became apparent that change was an important part of the results in this study. Change seemed to be connected to almost every other theme in the analysis, but since change was not really part of our research questions we decided to put change as a separate theme. And to use our other main theme names as the sub-themes for this theme.

4.4.1 Type Change

Initially, the notion of change appeared when interviewees at company A mentioned that they had changed VCS a few years ago. They had used a centralized VCS and opted for a distributed VCS instead. *"The main motivation for* [switching to] *Mercurial initially was the distributed system's ability of branching"*, according to the developer.

Likewise, both CMs at company B talked about changing the type of VCS used. One pointed out that there is a mentality difference between distributed and centralized VCSs. This means that all users of the VCS have to re-learn what they already know about version control, and the rhetorical question *"What is gained by changing?"* was posed by the CM. As many of the interviewees mentioned, it is important to weigh the advantages and disadvantages of potential changes. The other CM at company B said that newer projects are looking at Git as a possible replacement for Subversion, and that company B *"will, in my opinion, move to a distributed VCS as soon as possible (which, however, might take a while)"*.

Because company A is using a centralized way of working, all three interviewees from company A said that they would like to change back to a centralized VCS. The CM was a bit more nuanced and pointed out that if they were to change, the features associated with distributed VCSs would fall a long way down the list of desired features. However, a distributed VCS may still suit them because as a developer at company B said, distributed VCSs can work with centralized ways of working.

4.4.2 Process Change

Company B has a different set of CM strategies and processes for each of their projects. One of the projects has had an unspoken CM strategy but with the recent arrival of a CM to the project, a more formalized and official strategy has been adopted. Company A has changed CM processes and strategies a few times but *"the developers have been rather unaffected. Instead of using the Perforce tool to check out code, we write* hg clone *and then* hg pull *to get the code"*, as said by the developer.

After the change to Mercurial, the developer at company A noticed that *"projects increased exponentially, due to easier setup"*. This forced them to take a step back and reflect on the purpose of version control, and also on what the company was developing. After getting a more complete CM strategy everything went smoother, and as the developer said: *"Maybe it didn't have so much to do with the VCS, but it was still a consequence of it."*

5. DISCUSSION

In this section we discuss the points we found particularly interesting in our study.

5.1 Tool, Type and Process

As hypothesized in proposition 1, we expected the type to play a big role in the companies' choice of VCS. And that the choice would also be affected by their processes to a large extent. However, the results showed us quite the opposite; the tool has a larger affect than the type of VCS. We expected that if a company works co-located, even in quite large companies, a centralized VCS would fit their needs, and that a distributed VCS would have superfluous features that would not interest the company. A co-located company does not have the same need as an open source project, that value features like having clones of the whole repository locally, and making it easier for non-core developers to contribute[22]. But what about a company which has both co-located and distributed teams all working towards the same repository on a server in Sweden? Would they still feel that a centralized VCS fits their needs? This brings up other issues regarding the remote team members, and how they interact with the on-shore teams.

5.1.1 Centralized vs. Distributed

The challenges researchers have found[5] surrounding the use of centralized VCSs are for the most part not applicable in the corporate world. The first one (requiring network connection to work) could, however, still be relevant for companies with distributed developers. A good network connection was mentioned by company B as being a requirement for using a centralized VCS.

Could a company always be satisfied with a centralized system? A centralized system would provide for all their needs, especially a new and improved one with better merging and branching capabilities. The company would always have the latest code in the central repository, and project leaders could have a better overview of the project without having to check each developer's local repository. This point resonates with company A, who finds that their distributed VCS has excessive features for their co-located way of working. Company B, however, is looking into moving to Git for future projects.

If centralized VCSs provides everything that company B needs, why would they want to change? What they are lacking in their current VCS is some of the improved functionalities like branching, merging, and refactoring, that the newer distributed VCSs have. A change of VCS for company B would only be to solve drawbacks they currently see in their VCS, and not to get the distributed functionality.

Another interesting idea is the fact that distributed VCSs can be used in a centralized way, which one of the developers at company B mentioned. This is exemplified by company A, who uses Mercurial in a centralized way. It means that the type of the VCS would have less importance in the selection process. This could be a reason for why company B are looking at Git, even though they are not really geared towards a distributed system and are only after better tools.

Nevertheless, we still abide by the fact that, when using large amount of binary files, local complete copies of the repositories could still be a huge problem when using distributed VCSs.

5.1.2 Newer Tools are Better

Throughout the history of VCSs, the systems that have been popular have been those with the best tools, as one would expect. What is unexpected however, is that the VCS with best tools has often been the newer tool. In many other aspects of software development the best tools are usually those with a long history, and therefore have been improved upon for a long time and have reached stable versions. Among VCSs, new tools are often more technologically advanced, and also storing more metadata. We believe that this last point leads to innovation, and that this metadata is used when creating more tools[6] and plugins.

Contrary to this point is the fact that company A still uses the merge-tool from Perforce, even after they changed to use Mercurial as their primary VCS. They claim it is much better than the newer tool from Mercurial, and points to the easier GUI as the main reason.

5.1.3 The Silver Bullet

We acknowledge the fact that one solution will not work for everyone. Each situation is unique, and companies must analyze their needs when choosing a VCS. However, for the two companies in this case study, we believe that what they actually need is a new centralized system, with better functionality than today's systems. A centralized VCS works better for company A since a distributed VCS would still have the disadvantages of long checkout times of big repositories, as well as bad tools for the graphics file types company A uses. They also work in a completely centralized way.

It would also work for company B, since their problems are not related to the features of distributed VCSs; what they seek is better branching, merging and refactoring. A new

[5] See section 2.2 Centralized vs. Distributed

[6] It is not only the creators of VCSs that create tools.

centralized VCS would use current ideas and philosophies within VCSs, and improve upon them with features useful for larger companies.

5.2 Future of VCS

This leads us to the future of VCS within the corporate world. We believe the focus lies with increased functionality and usability. One of our interviewees believed that distributed VCSs was the future of VCSs, because of the open source community involvement. In his research article on the history of VCS, Ruparelia concurred in saying that *"One trend, however, is certain to persist: DVCS is becoming increasingly popular in the open source community and, over time, will replace centralized systems"* [24].

We, however, are more inclined to believe that the centralized systems still have an important role to fulfill, at least in the corporate world. But we also agree that distributed VCS are here to stay. To combine these two points, we believe in some kind of hybrid or umbrella solution that would use the best of both worlds.

5.2.1 Hybrid and Umbrella Systems

The hybrid or umbrella system is a very interesting evolution in VCSs and could be an alternative to the aforementioned idea of a new centralized VCS. Several of our interviewees also saw this as being the future of VCS. Perforce has developed a hybrid system, which they call Perforce Git Fusion [17]. It uses Perforce in the backend, while letting developers use the system like it is Git in the frontend [17].

An umbrella system could instead have two or several VCS backends, that would be coordinated to use a single frontend system. This way, the advantages of each system could be used to create a system that is greater than the sum of its parts.

Both of these ideas are based on improving the work flow for the developers and managers alike. For a developer that needs large graphics files to build and run the project, but is not interested in their complete history, only the latest version would be available in the local repository. At the same time, all versions of the graphics files would still be stored on the central server, and the artists that work on them still have access to all the history. The latest version of the code is always available at the central server, where the project manager can check the current status of the project.

5.2.2 Graphical User Interface and Visualization

The graphical user interface is very important, as many of our interviewees mentioned, in making the VCS more user-friendly and effective to use. This also agrees with what Candrlic et al. arrives at in their paper on desirable features of VCSs, who says that *"A major flaw of the basic version of the tool is the lack of a graphic user interface, and action triggering from the command line requires time to be mastered".* [3] A new or improved system would therefore be one with a more user friendly graphical user interface for visualizing branches, merging, conflicts, etc. for all the users, including the administrative role of the CM and the developers.

5.2.3 Integration

Another thing that would increase the usability of the VCS is integration, something that was mentioned at both companies. Integration would bring together other tools that are used throughout the lifecycle of the software project, like project management tools, bug tracking software, and IDE's. The idea behind this is that it increases usability, and makes it easier to visualize the whole project from one platform.

Integration is an important point that we think is very desired by the corporate world. Ruparielia says that a *"trend that we shall see continue into the future is for VCS to become increasingly integrated with: a) the entire software lifecycle, from requirements capture to defect tracking, and b) the broader configuration and change management tools and processes as defined by ITSM frameworks such as ITIL"* [24].

5.3 Change

What makes change interesting for this study, is why a company decides to change. Like de Alwis & Sillito elegantly puts it: *"The work involved in transitioning is significant and so we suppose that the reasons for switching must be compelling."* [8] One reason would be to make the work easier, as several of our interviewees from both companies said. Company B stated that they wanted to switch to a distributed VCS in the future, but is this really what they need? As they have already mentioned themselves, the distributed systems seem to have a steeper learning curve than the centralized system which they have now.

As was seen at company A, one positive thing about changing VCS is that the CM strategy gets brought into the spotlight. The CM strategy is something that needs to be worked on as much as any other aspect in a company.

So companies should ask themselves, *"what is gained by changing?"*

6. CONCLUSIONS

This study set out to investigate to what extent the type of VCS affects the choice of CM strategy at software development companies, and what VCS features are desirable for software development companies.

The study shows that our initial proposition about the type of VCS having a big impact on the process was turned on its head. Our results indicate that it is the tool that has the biggest effect on the process. Both companies in this study have chosen their VCSs based on the feature set of specific tools in comparison to other tools rather than the type. In addition to this, the type of VCSs they use is contrary to our first proposition. We therefore conclude that, as an answer to our first research question, the type of VCS only has a small effect on the CM strategy; and that we did not find any correlation between the way of working and the type of VCS used.

Our second research question focused on the desirable future features of VCSs. The majority of interviewees said that an effective graphical user interface, and especially visualization was something missing from current VCSs. Other features that we found were desired by these companies were atomic

commits, better merging capabilities, integration with other systems, and better handling of different file types.

An unexpected conclusion to this study was the emergence of a new centralized or hybrid VCS as a solution for what the companies in this study need from future VCSs. It would be interesting to see a wider feasibility study, to see if this solution would be useful for other companies as well.

We have in general come to the conclusion that companies have a need for other features than open source projects, often with focus on the administrative parts of VCSs. The learning curve, efficiency of daily operations, and ease of use are all important for a company selecting a new VCS.

6.1 Future Work

In future studies of this specific field of study we would like to see research that delve deeper into the following concepts.

Open Source
A comparative study between the needs of companies and open source projects, with regards to VCSs and CM strategies, would be an interesting compliment to this study. This would involve interviewing people both from the open source community and the corporate world.

Change
To look more into how the transition between VCSs affects a company's CM strategy would be of interest for all CMs who are exploring the possibility of replacing their current VCS. Change was found to be connected to all the other defined themes, but not relevant to our research questions. It would therefore be interesting to see a study where change and change management has a higher focus.

Hybrid
One interesting outcome of this study was the idea of a hybrid or umbrella VCSs. An investigation into existing hybrid VCSs, and requirements from software development companies would be useful in order to get a better understanding of this type of tool, and where the future could lead.

New Centralized VCS
Along the same lines as the hybrid VCS, it would be useful to do a feasibility study for a new centralized VCS. Can a new and improved centralized VCS compete with the rising popularity of distributed VCSs?

Industry
It would be interesting to look further into how the industry in which a company is working, be it gaming, medical, web, or any other industry, affects the choice of VCS. This would involve investigating if there are different needs in different sectors within the corporate software development field.

APPENDIX

A. INTERVIEW QUESTIONS

These questions are translated from the original Swedish versions, which was used during the interviews.

1. Describe in detail how you use and/or are affected by VCSs in your work. Please include details regarding for example branching and how collaboration is assisted or inhibited by the systems.
2. Do you have any strategy (explicit or implicit) regarding VCSs at your company? If so, please describe that strategy and also what VCS (or VCSs) you are using.
3. Do you believe that the type (centralized/distributed) of VCS you use are suitable for your way of working? What pros and cons do you see with centralized and distributed VCSs?
4. Do you think that the VCS affects your way of working? In what way, and to what extent? If you would change to a different VCS, would that choice be affected by your way of working?
5. What other VCSs do you have experience with? Which one do you think is best, and why?
6. Do you miss any functionality in the VCSs you use?
7. How would a future VCS look to you?

B. REFERENCES

[1] N. Bertino. Modern version control: creating an efficient development ecosystem. In *Proceedings of the 40th annual ACM SIGUCCS conference*, SIGUCCS '12, pages 219–222, New York, NY, USA, 2012. ACM.

[2] V. Braun and V. Clarke. Using thematic analysis in psychology. *Qualitative research in psychology*, 3(2):77–101, 2006.

[3] S. Candrlic, M. Pavlic, and P. Poscic. A comparison and the desireable features of version control tools. In *Information Technology Interfaces, 2007. ITI 2007. 29th International Conference on*, pages 121–126. IEEE, 2007.

[4] S. Chacon. *Pro Git*. Apress, 2009.

[5] D. Chudnov. Better, faster, stronger: Version control for everybody. *Computers in libraries*, 28(6):34–36, 2008.

[6] B. Collins-Sussman, B. Fitzpatrick, and M. Pilato. *Version Control with Subversion*. O'Reilly Media, 2007.

[7] J. W. Creswell. *Research design: Qualitative, quantitative, and mixed methods approaches*. SAGE Publications, Incorporated, 3rd edition, 2009.

[8] B. de Alwis and J. Sillito. Why are software projects moving from centralized to decentralized version control systems? In *Proceedings of the 2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering*, CHASE '09, pages 36–39, Washington, DC, USA, 2009. IEEE Computer Society.

[9] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian. Selecting empirical methods for software engineering research. In *Guide to advanced empirical software engineering*, pages 285–311. Springer, 2008.

[10] Eclipse Foundation. The open source developer report, 2011 eclipse community survey. `http://www.eclipse.org/org/community_survey/Eclipse_Survey_2011_Report.pdf`, 2011. Accessed: 2013-05-25.

[11] Eclipse Foundation. Results of eclipse community survey 2012. `http://www.eclipse.org/org/press-release/20120608_eclipsesurvey2012.php`, 2012. Accessed: 2013-06-02.

[12] S. Fish. Version control system comparison. `http://better-scm.shlomifish.org/comparison/comparison.html`, 2012. Accessed: 2013-05-14.

[13] D. Grune. Concurrent versions system, a method for independent cooperation. *Report IR-114, Vrije University, Amsterdam*, 1986.

[14] V. Jotov. An investigation on the approaches for version control systems. In *Proceedings of the 9th International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing*, CompSysTech '08, pages 73:V.11–73:1, New York, NY, USA, 2008. ACM.

[15] P. Krill. Torvalds's git: The 'it' technology for software version control. *InfoWorld.com*, Jul 26 2011.

[16] P. Louridas. Version control. *Software, IEEE*, 23(1):104–107, 2006.

[17] S. Merrill. Perforce aims to bring git to the enterprise. `http://techcrunch.com/2012/10/02/perforce-aims-to-bring-git-to-the-enterprise/`, 2012. Accessed: 2013-05-15.

[18] M. D. Myers and M. Newman. The qualitative interview in is research: Examining the craft. *Information and organization*, 17(1):2–26, 2007.

[19] B. O'Sullivan. Mercurial: The definitive guide. `http://hgbook.red-bean.com/read/`, 2009. Accessed: 2013-05-25.

[20] Paul. What are your pros and cons of git after having used it? `http://stackoverflow.com/questions/343675/what-are-your-pros-and-cons-of-git-after-having-used-it`, 2008. Accessed: 2013-06-01.

[21] M. Rochkind. The source code control system. *Software Engineering, IEEE Transactions on*, SE-1(4):364–370, 1975.

[22] C. Rodriguez-Bustos and J. Aponte. How distributed version control systems impact open source software projects. In *Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on*, pages 36–39. IEEE, 2012.

[23] P. Runeson and M. Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131–164, 2009.

[24] N. B. Ruparelia. The history of version control. *ACM SIGSOFT Software Engineering Notes*, 35(1):5–9, 2010.

[25] Spoike. Comparison between centralized and distributed version control systems. `http://stackoverflow.com/questions/111031/comparison-between-centralized-and-distributed-version-control-systems`, 2008. Accessed: 2013-06-01.

[26] A. N. Yurdusev. 'level of analysis' and 'unit of analysis': A case for distinction. *Millennium-Journal of International Studies*, 22(1):77–88, 1993.