Workflows and Distributed Version Control

Olof Johansson[1] and Daniel Persson[1]

Blekinge Institute of Technology
olof@ethup.se, daniel@silvertejp.org

Abstract. This bachelor thesis focuses on distributed version control systems and workflows used when working with such systems. The thesis will investigate benefits and disadvantages in using distributed version control in software development using a literature review of the available articles on the subject as well as a post-mortem analysis (with questionnaires and data collected from the build environment) of a student run software engineering project. We find that the migration costs are high, but that the advantages may outweigh its drawbacks for some. We conclude that many projects would benefit from migrating, but in particular, that new projects would not only benefit, but also not have high migration costs.

1 Introduction

We intend to investigate the adoption of the new trend within version control systems (VCS) — distributed version control systems (DVCS) — within the field of software engineering. The use of DVCS has increased in the last years, with the introduction of several implementations [5]. The move from centralized version control to DVCS may not always be unproblematic due to conceptual differences. It is important to investigate the benefits and challenges of DVCS in order to let software projects make informed decisions in their choice of, and use of, version control systems. The research will be conducted as a literature review, and as a post-mortem analysis of a student run software project. The paper will try to answer the following questions:

1. How do the developers adapt to a distributed workflow?
2. How does a distributed workflow affect code quality in relation to the release management and code review?

In Section 2 we introduce the concepts of VCS and give a brief history of its evolution. Our research methodology is described in Section 3. We continue with the findings from the literature review in Section 4. In Section 5 we introduce the student run Telia Smart Home project with focus on its use of version control systems. The primary means of doing the post-mortem analysis will be to conduct interviews with participants in their various roles. In addition to this we will also collect data (like build metadata) from the Telia Smart Home project management and configuration management tools and systems.

2 Background

Version control is one of the most valuable tools used in configuration management [2], especially important for software engineering projects with more than one developer. Here, we will provide a brief overview on the history of the concept and its uses — why it is used and how it is used.

We have decided to divide the history of Version Control Systems into three categories; the ones that only manages local files, the ones that rely on a central server to serve connecting clients, and finally the distributed ones, where every node can act as both a client and a server. The three categories roughly follow each other chronologically, naturally with transition-periods in between. At the moment the dominating tools are mostly client-server, but the distributed ones are rising fast in popularity, so perhaps we are currently in the start of a new transition.

2.1 The Local Era

The first VCS ever released was the *Source Code Control System*[1] (SCCS) in 1972. SCCS tried to solve a number of problems that software developers of that time had. Manually managing several versions of the same product simultaneously was not feasible in the long run for several reasons described by the creator of SCCS, Marc J. Rochkind [10]:

- The amount of space to store the source code may be several times that needed for any particular version.
- Fixes made to one version sometimes fail to get made to other versions.
- When changes occur it is difficult to tell exactly what changed and when.
- When a customer has a problem it is hard to figure out what version he has.

Instead of saving entire files in various states, SCCS stores the differences between versions of the same file (called diffs, or deltas [2]). With each delta SCCS also stores metadata such as who made the change, why, and when. This did not only save precious storage space, but also provided the *traceability* that the software developers had previously lacked [10].

It is worth noting that SCCS was not the only VCS that followed the philosophy of storing the files locally, *Revision Control System*[2] (RCS) which was essentially a wrapper around the UNIX^{TM} tool *diff* was released in 1982 and gradually started taking over as the dominant VCS for UNIX^{TM} [12]. RCS stored the latest version of a file and the deltas backwards towards the original state of the file, as opposed to SCCS that did the opposite, leading to faster access to the later revisions [9], which makes more sense from a user perspective.

[1] http://sccs.berlios.de
[2] http://www.gnu.org/software/rcs

2.2 The Client-Server Era

Although RCS was an improvement over SCCS it still had several flaws; it was still operating on single files only, and they were still stored locally, which meant that collaboration between multiple developers on the same project was cumbersome.

To be able to cooperate with his students, working on differing schedules, Dick Grune created shell scripts to centrally manage the project they were working on through RCS [6]. When the project was over he realized the potential and cleaned up the scripts and published them in late 1985. Two years later Brian Berliner rewrote the software in C, and Concurrent Versions System[3] (CVS) was born.

Much of the terminology we use today when we talk about version control was introduced with CVS. It also made branching a bit less cumbersome than it had been in RCS although merging the branches still required a lot of work.

With CVS in the front line and with proprietary version control systems starting to pop up in the early nineties the model of a central server that contained the repository that everyone worked in became the standard way to think about version control.

Although CVS simplified and popularized the use of version control systems, it has still received its fair share of criticism [9][1]: For example the revision numbers are created on a per file basis, which means that the latest revision number is not the same across the board which can sometimes be confusing. Other things like not handling symbolic links for example is claimed by the CVS developers to be carefully planned features (in the case of symbolic links due to security concerns) rather than bugs, but as the issues started to pile up Collabnet (founded by Tim O'Reilly that founded *O'Reilly Media* and Brian Behlendorf, co-founder of the Apache project) started the Subversion[4] project.

Subversion was released in 2000, and solved many of the issues that CVS had, while still maintaining the same working style of CVS. Subversion didn't really introduce any revolutionary ideas, but rather focused on the weaknesses of CVS like versioning of symbolic links, versioning for directories and file metadata and truly atomic commits (meaning if the transmission between client and server is interrupted, the commit is rolled back rather than processed half-way, much like transaction in a database [1] — in CVS this wasn't the case, and it could cause inconsistencies or corrupted data if it happened).

Other systems such as Perforce[5], IBM's ClearCase[6] and Microsoft's Visual SourceSafe[7] and Team Foundation Server[8] existed, but being proprietary tools they never grew popular in the open source community, and their popularity within big corporations is hard to measure. It should be noted that at least

[3] http://savannah.nongnu.org/projects/cvs
[4] http://subversion.apache.org
[5] http://www.perforce.com
[6] http://www-01.ibm.com/software/awdtools/clearcase
[7] http://msdn.microsoft.com/en-us/library/ms181038%28VS.80%29.aspx
[8] http://msdn.microsoft.com/en-us/vstudio/ff637362

Perforce has seen some use within open source, most notably Perl, prior to its migration to the DVCS system Git[9], described below.

2.3 The Distributed Era

The next innovation came from the open source community, where the centralized approach of CVS and Subversion didn't come as natural as it perhaps does in the corporate domain. Instead of storing the repository on a centralized server, distributed version control systems like GNU Arch[10], Monotone[11] and Mercurial[12] (hg[13]) stored a copy of the repository on every client that checked out a working copy. That way developers could always have access to the full history even without an internet connection.

One of the first distributed version control systems, the proprietary BitKeeper, infamously withdrew the rights for the Linux kernel project to freely use BitKeeper [9][11] which sparked Linus Torvalds to create a new, free tool: Git.

The main goal with git was to create a version control system that supported a BitKeeper-like workflow with high performance and strong integrity control. Since Linus also has a passionate hatred for CVS he has also said ironically that if there is any doubt in any design choice to be made, the opposite of the one CVS took is the way to go [7].

Git has in the past few years seen a steady growth in users and projects, and while the numbers compared to Subversion is still quite small, the change speed is pointing straight up [5].

3 Research Methodology

To answer the research questions proposed in the introduction we will conduct both a review of literature in the field of configuration management in general, and version control systems in particular, and then conduct a post-mortem analysis of the Telia Smart Home project. The primary means of doing the post-mortem analysis will be to conduct interviews with participants of the student projects in their various roles. In addition to this we will also collect data from various project management and configuration management tools and systems (e.g. Jenkins, Git or Redmine).

3.1 Literature Review Design

To provide a foundation for our thesis we will look at papers relating to centralized and decentralized version control, to give a clear view of what the differences

[9] http://git-scm.org
[10] http://savannah.gnu.org/projects/gnu-arch
[11] http://www.monotone.ca/main.php
[12] http://mercurial.selenic.com
[13] The "abbreviation" comes from the symbol for mercury in the periodic table.

between the two are, from a user perspective — i.e. in terms of migration costs, conceptual differences et cetera. Especially interesting is what kind of workflows that the DVCS tools enable, and compare it to the workflow used by the Telia Smart Home project.

After a preliminary survey of the available material on the topic we have noticed that scientific material regarding this is quite scarce, and we therefore choose not to provide any exact search strings, but instead decide to limit ourselves to the domains of configuration management in general, and version control systems in particular, and also provide a list of keywords that search-strings can be composed of:

- VCS, Version Control
- DVCS, Distributed Version Control
- SCM, Source Control Management
- Git, Subversion, Mercurial, Perforce, Darcs, CVS, BitKeeper
- Checkin, Checkout, Commit, Diff, Branch, Merge, Rebase
- Configuration Management
- Performance

We will focus our searches at Google Scholar and Elin, but if needed we will also resort to using for example IEEE Xplore or other relevant tools. If we are unable to find published academic papers on certain topics, we will also make use of white papers or other sources not published in academic journals (e.g. performance tests conducted by hackers and other non-academics).

The initial screening will be done by reading abstracts and conclusions to quickly dismiss irrelevant papers, and then go through the most promising candidates more thoroughly to decide what to use.

3.2 Post-mortem Analysis Design

The post-mortem analysis will primarily be based on interviews with participants in the Telia Smart Home project and questionnaires with members from other projects within the same course framework. We will base the interviews and questionnaires around these questions:

For the participants in the Telia Smart Home project we will ask the following questions:

1. How would you rate your proficiency in using Distributed Version Control Systems? (scale 1-5)
2. How would you rate your proficiency in using Version Control Systems? (scale 1-5)
3. Do you have any previous experience with VCS? If so, please describe.
4. What benefits do you see in using the workflow employed by the Telia Smart Home project?
5. What drawbacks do you see in using the workflow employed by the Telia Smart Home project?

6. Do you have any suggestions for improvements?
7. What is your overall opinion of the workflow?
8. Would you advice others to use the workflow? (yes/no)

For the participants in the other projects (within the same course framework) we will send out a subset of the survey as they do not employ the same workflow as the Telia Smart Home project:

1. How would you rate your proficiency in using Distributed Version Control Systems? (scale 1-5)
2. How would you rate your proficiency in using Version Control Systems? (scale 1-5)
3. What VCS software have you used before?
4. Do you have any previous experience with VCS? If so, please describe.
5. What Version control software are you currently using?
6. How do you work with your VCS tools?
7. Would you advice others to use your workflow? (yes/no)

The post-mortem analysis will also include an analysis of the metadata collected from the build environment, like number of build failures, their duration and ratio against build successes. The data collected from the project and configuration management tools will be used in combination with interviews and questionnaires to answer research question 2 about code quality.

4 Literature Review

4.1 Adoption

In these next few sections we will try to answer research question 1.

Migration Costs There a large number of open source as well as proprietary projects moving to one of the major distributed version control system; Perl, Curl and Parrot are projects that all has migrated to Git, from their earlier centralized VCSes[14]. The migrations has not always been cheap, e.g. the Perl migration took 22 months [3], so they must have had good reasons to do it. We will try to see why this might have been in the next few subsections.

A migration of VCS is twofold. There is migration of the repository, with moving every commit with its metadata and authorship information, every branch and every tag to the new VCS, making sure that nothing is lost and that nothing is broken. A problem in its own right, but with a lot of tools available producing generally good results. The other side, however, is migrating the developers and the workflows. Changes in interface design requires every developer to "relearn" the various actions and tools used. Something as trivial as commit IDs are different and has become a problem for some [3] — the centralized VCSes

[14] Perl migrated from Perforce, Curl from CVS and Parrot from Subversion.

commonly use incremental integers to identify a commit; DVCSes usually use a checksum of the content instead. The incrementing integers says more to a human when comparing two commits, which of the two commits are the newest, and perhaps gives a hint about how much newer [5]. The loss is an unfortunate effect of the distributed nature of DVCS. As every repository is "isolated" from each other, and only interacts when merging with each other, they can not know what numeric ID they should give to a commit to make it globally unique and still retain the incremental property. So instead checksums are used; well known algorithms are used to calculate a large value based on the contents of the commit. This difference between DVCS and VCS is one of the first a new DVCS user encounters.

Another change from centralized VCS is the decoupling of committing and merging. This makes it a two command operation to do the equivalent of:

```
svn commit
```

Looking past the short term impracticalities of having to do two commands to merge with a central server, it gives the developer freedom to not merge at all, or not merge "right now". Perhaps it is not even possible to merge — as is the case in airplanes or being in remote areas where no network connectivity is available or if the developer does not have write access to the upstream repository.

Another change, not a change in the interface but a change of VCS nomenclature, is perhaps an even bigger obstacle for migration [5]. Some terms are introduced, some terms are removed and some terms are redefined. Committing is, as described, no longer publishing your changes in the central repository. Pulling and pushing are common terms within DVCS for things that don't even exist in a centralized VCS. In short, there will have to be a lot of documentation of procedures, tools and workflows when migrating to a new VCS — especially a DVCS [3].

Architecture As mentioned in the section on migration costs, the fundamental differences in the distributed version control architecture vis a vis the centralized ditto makes it hard to transfer workflows and techniques directly. The differences do make it possible to do other things and brings a number of benefits together with its disadvantages.

The distributed nature means that you have certain limitations on what can be done to enforce synchronization: e.g. there is no file locking support (making sure that only one developer works on a certain file at a time) [4] and it's harder to impose automated content control. It makes it hard to delete history — this would have been problematic for e.g. the FreeBSD developers when they were forced by a court decision to cease distribution of certain older components [3]. But by the same token, it works as an implicit distributed backup system, where each checked out repository is a repository in its own right. Data loss at a single node won't be a critical issue, as the complete repository with content and history is replicated numerous times [3]. Another benefit from the distributed architecture of DVCS is the ability to work independently, isolated from the central repository. With this, a developer could work when traveling to or from remote offices or conferences [3][9].

The research done in the field has largely depended on the activities of various open source projects, with its wealth of openly accessible data and discussions [3][5]. These kind of projects has some specific problems and needs, some which only has a limited effect on non open source projects. One positive effect of distributed version control systems is that it lowers the bar for new contributors. They can easily fork a project and commit changes, and then make a request to be merged into the mainline [3]. This limits the phenomenon known as ”commit whoring” or ”commit politics”, where people try to get commit access just for the sake of it (as a status symbol).

Performance The fact that anyone using a DVCS is working with a local repository has big impacts on performance. In a centralized VCS every operation requires access to the central server, and thus adds network latency to every operation [7]. Additionally, every developer that works towards the same central server adds more to its workload, making it scale quite badly [8]. In contrast, DVCSes do not work towards the same server (although some repositories are perhaps considered ”more important” than others) and thus scales very well with almost any amount of developers [11]. Working locally leads to operations like branching and merging being much faster and smoother, encouraging developers to use them more [3].

4.2 Quality Implications

In this section we will try to answer research question 2.

The fact that most operations are so fast in DVCSes will have an impact on how people work with their SCM tools. The fact that branching is such a cheap operation will encourage developers to actually use branches where they see fit [3], without taking things like workload on a central server, namespace pollution, or wait times into consideration. Coupled with the fact that merging branches is relatively painless [3][4] — and thus resulting in merging being done when it should, instead of being postponed to delay the suffering for the developer, creating unnecessary conflicts and more suffering [4] — will in turn lead to a more thorough and in a sense more correct use of the VCS, resulting in a better history and potentially better code.

Another powerful tool that can be used more frequently due to the huge performance-benefits that DVCSes provide is `bisect`. It is very easy to use, but can prove to be very useful in finding bugs, or rather finding out when they are introduced [4]. What `bisect` does is basically a binary search between two revisions (between which you know that a bug is introduced) by checking out revisions and asking the developer if the bug is present or not. Provided that the bug can be replicated with for example a script, this task can be automated fairly easy, and the fact that everything runs locally keeps the performance up, and prevents the network from being congested. Needless to say, when looking for a particular bug it is a huge advantage to know the exact commit that introduced it.

Since the primary mean of sharing and spreading code in a DVCS workflow is by pulling other peoples changes into your local repository, introducing quality practices like *code reviews* fits in very well. Whenever anyone pull code from someone else, they simply look at the diff, and if it does not meet the agreed upon quality standards, they don't merge them until it does [4].

5 The Telia Smart Home Project Context

The last semester of the Software Engineering program at Blekinge Institute of Technology is concluded with a large group software engineering project. Within this course, the students are put in charge of themselves organizing (in terms of methodology and tools) and executing a software project for a customer from the industry. The authors of this paper participated in one such project, *Telia Smart Home*. The project team consisted of eight students working full time.

When choosing a VCS the project members had a number of considerations to account for. VCS can be used in a number of ways. Some limits are posed by the choice of tools. Subversion is built for centralized version control, whereas tools like git or mercurial are built for decentralized. Members of the latter category are generally more flexible; e.g. git is almost as simple to run centralized as running it decentralized as it is intended. The Telia Smart Home project has chosen to use git. The reasons for this are several: the ability to use git decentralized as well a centralized was a major point. Also, the prior experience from some members of the project made the transition for other members less burdensome.

But defining a workflow does not end with selecting a tool. As stated, git is very flexible in the way developers interact with it. It is not uncommon to use it centralized, and in fact, the Telia Smart Home project did just this in its documentation repository. But one of the key strength is its decentralization. It has no enforced ”central repository” through which all changesets must pass. If the team want to have a centralized repository it must itself give an arbitrary repository that semantic meaning.

Are there any alternatives to be considered? Certainly. Again, git, as a DVCS, is indeed quite flexible. It will yield to the wishes of the user. There are a number of easily recognized workflows (perhaps not always mutually exclusive). Some of the most popular ones are presented here:

– **Centralized**: Using git in a way similar to e.g. Subversion or other more traditional tools still give the user some advantages. The process of branching and merging is many times more well developed in git than it is in tools like Subversion. A property of the distributed nature of git gives yet another benefit: having all of the history available locally. You don't need network to be able to see what changes was introduced to a particular commit, and indeed no network to be able to see the commit log.

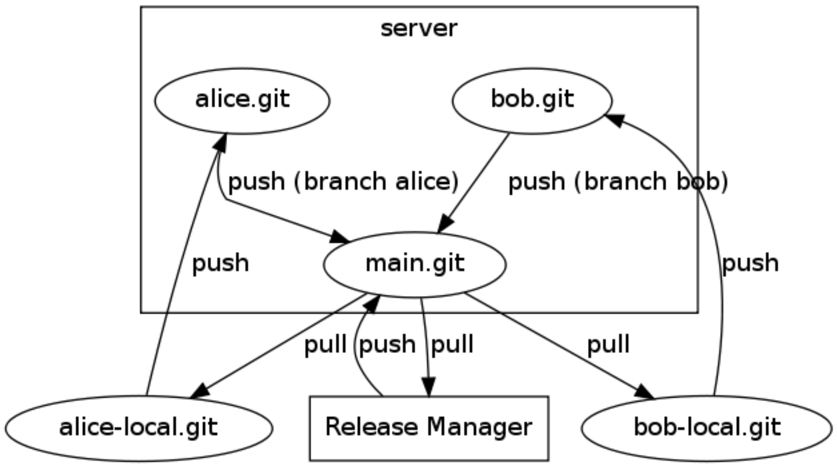


Fig. 1: Workflow description

- **Topic branches**: Working on a new cool, but experimental feature? Perhaps it is not as tested as the rest of the system is. You probably don't want to have it in the main code. Create a branch specifically for this "topic" or "feature". When it is done, it may get merged into the mainline branch.
- **Benevolent dictator**: A workflow, especially popular within open source projects, is the benevolent dictator workflow. There is one designated maintainer, and other contributors make "pull requests", primarily via e-mail. If the benevolent dictator accepts the patch, he or she merges it into his or her own repository, where everybody gets their source code from. The name, "Benevolent dictator", is a term jokingly used about Linus Torvalds, the initial author and later primary maintainer for the Linux kernel (and also, the initial author of git!).

Each of these workflows are appropriate in some situations. However, none of these totally fitted the needs of the Telia Smart Home project. Instead, the Telia Smart Home project designed its own workflow (see Figure 1). The workflow used for the code repositories in the project is based upon individual developer repositories and branches. Each developer has their own repository, and makes commits to this. The system (with the help of so called hooks (i.e. scripts triggered by specific events)) then pushes changes to a repository on a central build server. Here, a build is triggered using the *Jenkins* continuous integration system and the commit is also forwarded to the so called "baseline" repository, but only to a developer's private branch. It is not merged into the actual baseline code, but it is still accessible for other developers.

The release manager is now notified that there is a commit awaiting approval. He or she can check build and test status on Jenkins and see the delta between baseline and the proposed patch. After possibly running local tests, the release manager can either approve and merge the commit or reject it, and in the latter case inform the author of why it wasn't suitable for inclusion. If the commit is merged, other developers are now encouraged to merge it into their local repositories, their working copy of the source code.

When designing this system, the sought benefits was the integrated code review and approval system. With this, the release managers could validate adherence to coding conventions as well as making sure that the system is buildable and all unit tests passes. Possibly, much of this could even have been automated (e.g. static code analysis and coding style validation tools are commonplace) to an even higher degree, but that is, as always, a cost/benefit question. It is innovative and unique, but it does build upon the so called "benevolent dictator" workflow, especially the release management component of it. The success of this workflow from high-profile projects like the Linux kernel has been an inspiration for the design. The reason why the project rejected the topic branch workflow at an early stage was the problems associated with topic branches in the early stages of a software project, where a lot of interdependencies exist and basic infrastructure is needed. In later stages of the project, this workflow would have been more feasible, but the overhead associated with switching workflow was deemed too high. It has been used in some rare cases, where code was at a very experimental stage.

6 Results from Post-mortem Analysis

Regarding the workflow employed by the Telia Smart Home project, the survey conducted resulted in generally positive feedback (see appendix B), with only some minor reservations. Some concern was raised regarding the way the project "abused git" in the way branches were automatically re-pushed to branches in the mainline repository. A problem with the release manager approach was that it easily becomes a bottleneck, if for example the release manager is absent or busy. Several survey participants claimed to have experienced this problem. Another survey participant noted that the workflow looked more complex than it really was. Other participants noted that the code review came natural and worked well, and that the mainline build was rarely broken.

Comparing to the data collected from the continuous integration environment (Jenkins) (see Appendix C), we can easily see some indications of the project quality state:

- The number of successful builds represents 86.85% of all builds. See Figure 2.
- The average time it took to fix a build failure[15] was 30 minutes, with a mean time of 21 minutes. See Table 1.
- There were no build errors after 2011-03-30. See Table 1.

[15] A build failure is defined as either a compilation error, one or more failing unit tests or problems in the build environment.

Table 1: Statistics for build failures

Date	Build attempts	Duration (min)
2011-03-10	2	5
2011-03-10	3	21
2011-03-11	1	49
2011-03-11	1	4
2011-03-14	1	25
2011-03-30	11	115

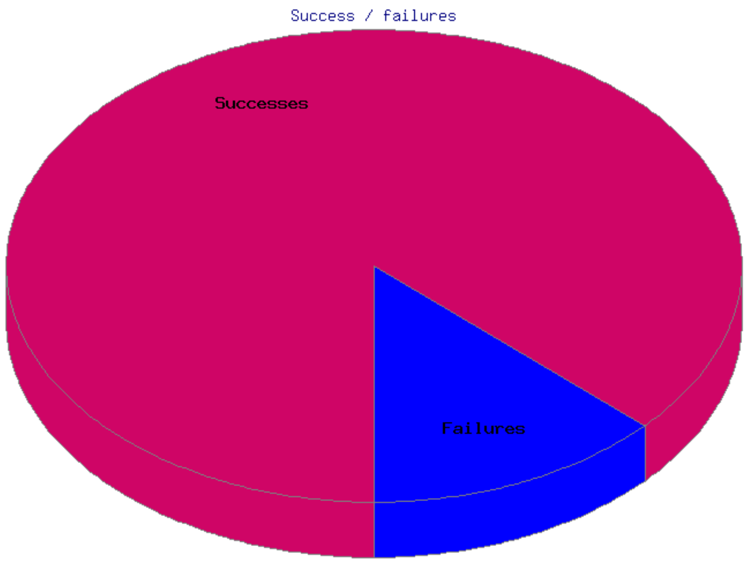


Fig. 2: The percentage of successful builds (red) and failed builds (blue) within the Telia Smart Home project, over a period of 153 builds.

It is hard to know what kind of impact the release management approach used by the project had without control data, but the situation was satisfactory for the project. The fact that build was only broken in the beginning of the project could be an indication that the release management was effective once the team and release managers got used to the workflow.

When looking at the survey results from the participants in other projects, the general impression is that little thought has been put into the design of their VCS workflows. Only a minority of the respondents "understood" what *workflow* meant, in the VCS context. Those that did, described a simple centralized workflow — with or without DVCS tools.

Another observation was that almost half of the respondents had never used a version control system until the current project. Those that did have prior experience had used primarily Git or Subversion, with some minor use of Mercurial also accounted for. The respondents stated that they had prior experience with Mercurial did not report that they had used Git. See Figure 3a. Most of the respondents described their current proficiency level with VCS in general as a "three" on a scale from one to five. The DVCS proficiency was centered around three and four on the same scale. See Figure 3b.

6.1 Analysis of Post-Mortem Findings

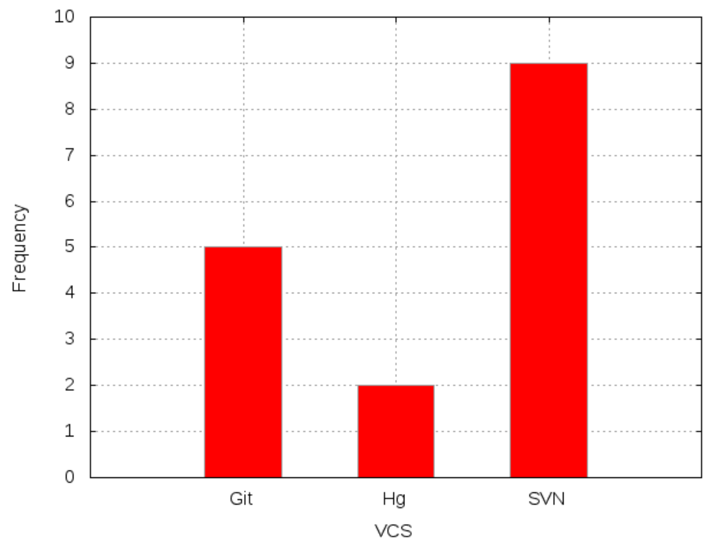
The survey responses within the Telia Smart Home project made it clear to us that the workflow was feasible and that the developers unanimously was happy with it, albeit with some minor criticism.

The re-pushing of branches could very well have been excluded from the workflow; the reasons for having it was to ease the work for release managers by having all changes in one repository and not spread out. The other reason was to allow easy overview in the issue tracking system (Redmine) which was limited to only show one repository.

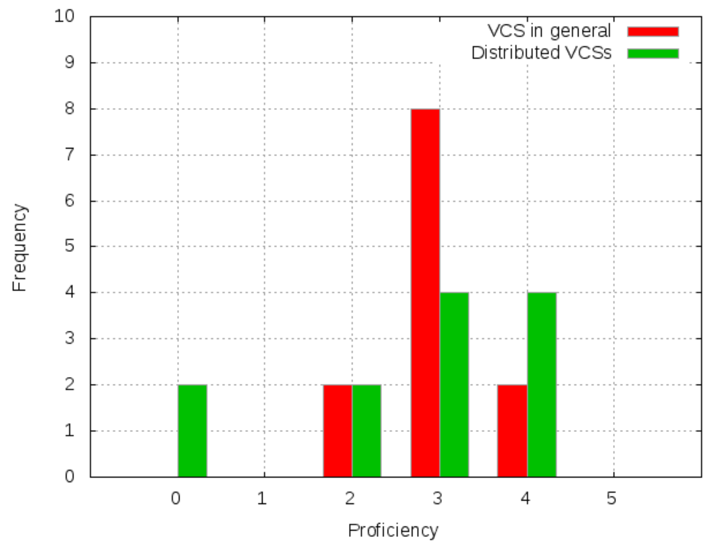
The release management criticism is not in any way limited to the Telia Smart Home workflow, it is present in any workflow using a review process to accept and merge commits. One possible solution could be for example having a dedicated person for the review process, but in this project the cost would have been too large. Sharing the responsibility of release management between several people could also remove the downtime of the review process when the release manager is absent. Overall the release management and code review has worked well, with few hick-ups, and as Table 1 shows, the mainline build was broken relatively few times, and only for short periods of time.

7 Discussion

The literature review focused on migration costs, whereas the post-mortem analysis investigated the adoption of DVCS in a new project. The migration in the latter case was an individual adoption rather than the "collective" project migration usually described in the literature. Still, some of the conclusions clash



(a) Previous experience with VCS tools



(b) Proficiency with VCS and DVCS

Fig. 3: Results from survey with external project participants

with the findings in the literature review. The papers dealing with migrations emphasize the problems with individual developers having to adopt to new interfaces and changed behavior. During the course of the project, this has been in large a non-issue. We suspect that we would find other results if we would enlarge the scope of the research and include a more diverse target group with more experience of software engineering and version control.

The Telia Smart Home workflow was enabled by, and requires, DVCS. It would not have been possible to do this with, e.g., Subversion or CVS. Our use of branching and merging was a core part of the workflow. This confirms what we found in the literature review, branching and merging is more often a problem and a burden in centralized VCS than it is in DVCS.

8 Conclusion

Our findings suggest that a project using Subversion or another centralized VCS could benefit from migrating to a DVCS — the migration may not come cheap, but the benefits was enough to convince e.g. Perl, Parrot and Curl to change to Git. The benefits in terms of more flexible workflow opportunities, intuitive integration of release management and the low cost of merging and branching all work in favor of the DVCSes. Its up to every project to determine whether or not the benefits outweighs the migration costs.

One type of cases that should be especially motivated to evaluate the use of DVCS are new projects, without any prior processes, routines and infrastructure. The drawbacks are few and mostly minor; the most severe drawback is personal preferences. Some are used to working with a specific tool or centralized VCS in general and may be more productive in a familiar setting. This should be taken into consideration when evaluating the choice of a VCS.

References

1. Apache subversion features. http://subversion.apache.org/features.html.
2. V. Ambriola, L. Bendix, and P. Ciancarini. The evolution of configuration management and version control. *Software Engineering Journal*, 5(6):303–310, 1990.
3. Brian de Alwis and Jonathan Sillito. Why are Software Projects Moving From Centralized to Decentralized Version Control Systems? *ICSE Workshop on Cooperative and Human Aspects on Software Engineering*, 2009.
4. Brian O'Sullivan. Making Sense of Revision-Control Systems. *Communications of the ACM*, 2009.
5. Christian Bird and Peter C. Rigby and Earl T. Barr and David J. Hamilton and Daniel M. German and Prem Devanbu. The Promises and Perils of Mining Git. *6th IEEE International Working Conference on Mining Software Repositories*, 2009.
6. Grune, D. Concurrent Versions System, a method for independent cooperation. *Report IR-114, Vrije University, Amsterdam*, 1986.
7. Linus Torvalds. Linus Torvalds on Git. Transcript for Google Tech Talk. http://git.or.cz/gitwiki/LinusTalk200705Transcript, 2007.

8. Matt Mackall. Towards a Better SCM: Revlog and Mercurial. *Linux Symposium*, 2006.
9. Ollivier Robert. DVCS or a new way to use Version Control Systems for FreeBSD. 2006.
10. Marc J. Rochkind. The source code control system. *IEEE Transactions on Software Engineering*, SE-1 No. 4, 4 December 1975.
11. Maha Shaikh and Tony Cornford. Version Management Tools: CVS to BK in the Linux Kernel. *3rd Workshop on Open Source Software Engineering*, 2002.
12. W.F. Tichy. RCS — a system for version control. *Software: Practice and Experience*, 15(7):637–654, 1985.

Appendices

A Data from Literature Review

Articles passing the initial relevance screening:

A.1 Version Management Tools: CVS to BK in the Linux Kernel

An article by M. Shaikh and T. Cornford, written in 2002 and published in 3rd Workshop on Open Source Software Engineering.

> **Abstract.** Version management tools might be seen as a prerequisite for open source development today as projects become too large to be managed by maintainers alone. Yet the OS[16] process depends on fluid coordination and collaboration with the underlying qualities of this process based on firm trust and respect for fellow developers. This paper is a study of how debate over version tools reflects governance and decision making in an OS community. The paper is based on a study of the Linux kernel community as it first saw a partial acceptance of the CVS tool, and then later adopted BK[17]. The paper explains the adoption process in relation to governance concerns, license issues, and questions of technical performance.

The described problems with introducing a distributed version control system into a such a large project as the Linux kernel, may give us insight into the adaption process. This paper relates to research question 1.

A.2 Making Sense of Revision-Control Systems

An article by B. O'Sullivan, written in 2009 and published in Communications of the ACM.

> **Abstract.** All revision-control systems come with complicated sets of trade-offs. How do you find the best match between tool and team?

A high-level description of revision control tools and concepts and its use will give us a reference on what properties is interesting in version control workflows.

[16] OS, as in Open Source. Not to be confused with Operating System, our comment.
[17] Bitkeeper, a proprietary version control system, our comment.

A.3 Darcs: Distributed Version Management in Haskell

An article by D. Roundy, written in 2005 and published in Proceedings of the 2005 ACM SIGPLAN workshop on Haskell.

Abstract. A common reaction from people who hear about darcs, the source control system I created, is that I sounds like a great tool, but it is a shame that it is written in Haskell. People think that because darcs is written in Haskell it will be a slow memory hog with very few contributors to the project. I will give a somewhat historical overview of my experiences with the Haskell language, libraries and tools.
I will begin with a brief overview of the darcs advanced revision control system, how it works and how it differs form other version control systems. Then I will go through various problems and successes I have had in using the Haskell language and libraries in darcs, roughly in the order I encountered them. In the process I will give a bit of a tour through the darcs source code. In each case, I will tell about the problem I wanted to solve, what I tried, how it worked, and how it might have worked better (if that is possible).

A case study into a specific distributed version control system. Even though the paper deals in large parts with the implementation details, the actual usage and interaction with the system is described. This paper relates to research question 1.

A.4 Towards a Better SCM: Revlog and Mercurial

An article by M. Mackall, written in 2006 and published in the Linux Symposium.

Abstract. Large projects need scalable, performant, and robust software configuration management systems. If common revision control operations are not cheap, they present a large barrier to proper software engineering practice. This paper will investigate the theoretical limits on SCM performance, and examines how existing systems fall short of those ideals.
I then describe the Revlog data storage scheme created for the Mercurial SCM. The Revlog scheme allows all common SCM operations to be performed in near-optimal time, while providing excellent compression and robustness.
Finally, I look at how a full distributed SCM (Mercurial) is built on top of the Revlog scheme, some of the pitfalls we've surmounted in on-disk layout and I/O performance and the protocols used to efficiently communicate between repositories.

Yet another case study into a specific distributed version control system. Again, this also deals with technical details, but the interaction and usage pattern is interesting for our research. This paper relates to research question 1.

A.5 The Promises and Perils of Mining Git

An article by C Bird, P. C. Rigby, E. T. Barr, D. J. Hamilton, D. M. German and P Devanbu, written in 2009 and published in 2009 6th IEEE International Working Conference on Mining Software Repositories.

> **Abstract.** We are now witnessing the rapid growth of decentralized source code management (DSCM) systems, in which every developer has her own repository. DCSMs facilitate a style of collaboration in which work output can flow sideways (and privately) between collaborators, rather than always up and down (and publicly) via a central repository. Decentralization comes with both the promise of new data and the peril of its misinterpretation. We focus on git, a very popular DSCM used in high-profile projects. Decentralization, and other features of git, such as automatically recorded contributor attribution, lead to richer content histories, giving rise to new questions such as "How do contributions flow between developers to the official project repository?" However, there are pitfalls. Commits may be reordered, deleted, or edited as they move between repositories. The semantics of terms common to SCMs and DSCMs sometimes differ markedly, potentially creating confusion. For example, a commit is immediately visible to all developers in centralized SCMs, but not in DSCMs. Our goal is to help researchers interested in DSCMs avoid these and other perils when mining and analyzing git data.

And a third (and final) case study into a specific version control system. This paper does not primarily deal with technical details however, and instead focusing on interaction details. This paper relates to research question 1.

A.6 Why Are Software Projects Moving From Centralized to Decentralized Version Control Systems?

An article by D. de Alwis and J. Sillito, written in 2009 and published in 2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering.

> **Abstract.** Version control systems are essential for co-ordinating work on a software project. A number of open- and closed-source projects are proposing to move, or have already moved, their source code repositories from a centralized version control system (CVCS) to a decentralized version control system (DVCS). In this paper we summarize the differences between a CVCS and a DVCS, and describe some of the rationales and perceived benefits offered by projects to justify the transition.

This paper is relevant to us and important for us to understand the reasons for migration, and as a reference for comparisons with centralized version control systems. This paper relates to research question 1.

A.7 DVCS or a new way to use Version Control Systems for FreeBSD

An article by O. Robert, written in 2006.

Abstract. FreeBSD, like many open source projects, uses CVS as its main versioon control system (VCS), which is an extended history of all modifications made since the beginning of the project in 1993. CVS is a cornerstone of FreeBSD in two ways: not only does it record the history of the project, but it is a fundamental tool for coordinating the development of the FreeBSD operating system.
CVS is built around the concept of centralised repository, which has a number of limitations.
Recently, a new type of VCS has arisen: Distributed VCS, one of the first being BK from BitMover, Inc. Better known from the controversy it generated when Linus Torvalds started using it, it has nonetheless changed the way some people develop software.
This paper explores the area of distributed VCS. We analyse two of them (Arch in its Bazaar incarnation and Mercuriel and try to show how such a tool could help further FreeBSD development, both as a tool and as a new development process.[18]

A paper dealing with a potential migration from a centralized version control system (CVS), to a decentralized. What would be the benefits for the FreeBSD project? What would be the drawbacks? This paper relates to research question 1.

A.8 BitKeeper for Kernel Developers

An article by V. Henson and J. Garzik, written in 2002 and published in Ottawa Linux Symposium.

Abstract. BitKeeper is a revolutionary new distributed source control management suite which is ideal for Linux kernel development. BitKeeper provides tools which automate and simplify many common kernel development tasks. In this paper, we describe basic BitKeeper concepts and operations, BitKeeper solutions for common kernel development problems, and a workflow for interacting with other Linux developers using BitKeeper. We also discuss some of BitKeeper's shortcomings and what is being done to correct them. We conclude that BitKeeper can dramatically imrpove the efficiency of Linux kernel developers.

Bitkeeper is a distributed version control system, and was for a long time used by the Linux kernel developers for their version control needs. Even though it was surrounded with controversy, due to its non-free, proprietary status and hostility against reverse engineering, it was still a major breakthrough for distributed version control. This paper relates to research question 1.

[18] *Sic*, in regards to unbalanced parentheses

A.9 Software Configuration Management: A Roadmap

An article by J. Estublier, written in 2000 and published in Proceedings of the conference on The future of Software engineering.

Abstract. This paper, in the first chapter summarizes the state of the art in SCM[19], showing the evolution along the last 25 years. Chapter 2 shows the current research work under way in the area. In chapter 3, the challenges SCM has to take up, as well as SCM future research are discussed.

This paper is a general overview of the configuration management field. It is useful as a basis and context for our research into a specific configuration management concept and technology, namely version control. This paper relates to research question 1.

A.10 Learning by Doing: Introducing Version Control as a Way to Manage Student Assignments

An article by K. L. Reid and G. V. Wilson, written in 2005 and published in ACM SIGCSE Bulletin.

Abstract. Professional software developers use version control systems to coordinate their work, and to provide an unwindable history of their project's evolution. In contrast, students in most programming courses use a homegrown electronic submission program to submit their work, and email to coordinate with partners when doing team projects. In May 2003, we began using CVS, a popular open source version control system, as an assignment submission system. Students receive starter code by checking out each student's repository, and committing the marks. Our experience to date shows that this is both a simpler and more flexible way to manage student assignments, and also an excellent way to teach them how to use a fundamental software development tool.

As a large part of our research will be about adaptability and ability to learn and understand our workflow, it is interesting to see other experiences within version control education. How did the students adapt to this somewhat unorthodox assignment submission system? This paper relates to research question 1.

[19] SCM, as in Software Configuration Management. Not to be confused with Source Control Management.

A.11 RCS — A System for Version Control

An article by W.F. Tichy written in 1985 and published in Software: Practice and Experience.

Abstract. An important problem in program development and maintenance is version control, i.e., the task of keeping a software system consisting of many versions and configurations well organized. The Revision Control System (RCS) is a software tool that assists with that task. RCS manages revisions of text documents, in particular source programs, documentation, and test data. It automates the storing, retrieval, logging and identification of revisions, and it provides selection mechanisms for composing configurations. This paper introduces basic version control concepts and discusses the practice of version control using RCS. For conserving space, RCS stores deltas, i.e., differences between successive revisions. Several delta storage methods are discussed. Usage statistics show that RCSs delta storage method is space and time efficient. The paper concludes with a detailed survey of version control tools.

This paper thoroughly describes RCS, and how it is designed. It is important to understand basic concepts of this to answer both research questions.

A.12 Concurrent Versions System, A Method for Independent Cooperation

An article D. Grune written in 1986 and published in Report IR-114, Vrije University.

Abstract. People working together on a set of files in real-world circumstances may often interfere with each other. To avoid such interference, the idea of an abstract data type can be used. The abstract data type introduced here consists of a single repository, containing versions of the files, together with a (small) number of access routines. Each participant has his own copy of a set of files as represented by the repository, and uses essentially two access routines, one to merge into his copy changes made to the repository by others, and one to merge changes in his copy into the repository. (There are several other access routines, for listing contents, examining differences, etc.) The abstraction is not perfect, but violations (conflicts) are generally detected and reported.
The implementation of the abstract data type "repository" starts from version control primitives that can handle a single file and then coordinates their actions. All access routines are programmed as UNIX shell scripts and use the RCS programs as version control primitives. In programming the access routines, many more possibilities had to be taken into account than was intuitively reasonable. Examples are given.

Similar to the RCS paper, this paper is about CVS, and is also written by its creator. With CVS being a major influence on most modern VCSes, this is also imperative for the understanding of basic version control concepts. This paper relates to research question 1.

A.13 Software quality: the elusive target

An article by B. Kitchenham, and S.L. Pfleeger, written in 1996 and published in IEEE Software.

> **Abstract.** If you are a software developer,manager or maintainer, quality is often on your mind. But what do you really mean by software quality? Is your definition adequate? Is the software you produce better or worse than you would like it to be? In this special issue, we put software quality on trial, examining both the definition and evaluation of our software products and processes.

To be able to measure how software quality is affected we need some way of measuring software quality, this paper hopefully gives a good foundation for us to find a suitable metric. This paper relates to research question 2.

A.14 A practical view of software measurement and implementation experiences within Motorola

An article by M.K. Daskalantonakis, written in 1992 and published in IEEE Transactions on Software Engineering.

> **Abstract.** The purpose of this paper is to describe a practical view of software measurement that formed the basis for a company-wide software metrics initiative within Motorola. A multi-dimensional view of measurement is provided by identi- fying different dimensions (e.g., metric usefulness/utility, metric types or categories, metric audiences, etc.) that were considered in this company-wide metrics implementation process. The defi- nitions of the common set of Motorola sofiware metrics, as well as the charts used for presenting these metrics, are included. The metrics were derived using the GoaVQuestiodMetric approach to measurement. The paper distinguishes between the use of metrics for process improvement over time across projects and the use of metrics for in-process project control. Important experiences in implementing the software metrics initiative within Motorola are also included.

Another paper dealing with software quality; dealing more with classifications of bugs et cetera this could be useful when analyzing quality related data. This paper relates to research question 2.

A.15 The Evolution of Configuration Management and Version Control

An article by V. Ambriola, L. Bendix and P. Ciancarini, written in 1990 and published in Software Engineering Journal.

Abstract. The activities of configuration management and version control are common to a number of engineering tasks. These activities are particularly important for software engineers, since during most of a system lifecycle they have to deal with a growing number of versions of a single component, and to rebuild the complete system in different ways using different components. These tasks are repetitive and trivial, and they require a lot of manual work and accuracy. In this paper, we show how the problem of automating these activities has been solved in a number of software development environments. We describe the evolution of systems for configuration management and version control from simple stand-alone tools, such as make and SCCS (based on an underlying file system), towards more integrated systems based on a project database.

This paper gives a more abstract view of Configuration Management in general, as well as an overview of older VCSes. This relates to research question 1.

B Questionnaire Results

B.1 Internal Surveys

Please rate your proficiency using the following technologies:

VCSs in general	4
Distributed VCSs	4
Programming in general	5
Do you have any previous experience with VCSs? If so, please describe.	
What benefits do you see in using the workflow employed by the TSH project?	Everyone is kept up to speed with the progress all the time
What drawbacks do you see in using the workflow employed by the TSH project?	None that i can think of
Do you have any suggestions for improvements?	No
What is your overall impression of the workflow?	Very good
Would you advice others to use the workflow?	Yes

Please rate your proficiency using the following technologies:

VCSs in general	3
Distributed VCSs	3
Programming in general	4
Do you have any previous experience with VCSs? If so, please describe.	Used Git in a couple of projects
What benefits do you see in using the workflow employed by the TSH project?	Ensures a buildable product at all times; clean, consistent code
What drawbacks do you see in using the workflow employed by the TSH project?	
Do you have any suggestions for improvements?	Possibly change release manager once in a while
What is your overall impression of the workflow?	Works good, except for the occasional nagging needed to get changes merged by the release manager
Would you advice others to use the workflow?	Yes

Please rate your proficiency using the following technologies:

VCSs in general	2
Distributed VCSs	2
Programming in general	4
Do you have any previous experience with VCSs? If so, please describe.	No
What benefits do you see in using the workflow employed by the TSH project?	Makes a good flow
What drawbacks do you see in using the workflow employed by the TSH project?	None
Do you have any suggestions for improvements?	No
What is your overall impression of the workflow?	Good
Would you advice others to use the workflow?	Yes

Please rate your proficiency using the following technologies:

VCSs in general	1
Distributed VCSs	1
Programming in general	2
Do you have any previous experience with VCSs? If so, please describe.	No
What benefits do you see in using the workflow employed by the TSH project?	I can make mistakes and there is no problem
What drawbacks do you see in using the workflow employed by the TSH project?	I have to call someone to get help, at least in the beginning of the project
Do you have any suggestions for improvements?	No
What is your overall impression of the workflow?	I think it worked great
Would you advice others to use the workflow?	Yes

Please rate your proficiency using the following technologies:

VCSs in general	2
Distributed VCSs	2
Programming in general	3
Do you have any previous experience with VCSs? If so, please describe.	Gallega project, setting up personal VCS (both Subversion)
What benefits do you see in using the workflow employed by the TSH project?	Main build is not broken, code is reviewed
What drawbacks do you see in using the workflow employed by the TSH project?	Workflow is disturbed when release managers are absent
Do you have any suggestions for improvements?	No
What is your overall impression of the workflow?	It worked fine
Would you advice others to use the workflow?	Yes

Please rate your proficiency using the following technologies:

VCSs in general	4
Distributed VCSs	4
Programming in general	4
Do you have any previous experience with VCSs? If so, please describe.	Yes, with Git and Subversion
What benefits do you see in using the workflow employed by the TSH project?	Code is reviewed. Release managers work as gatekeepers and does not merge bad code
What drawbacks do you see in using the workflow employed by the TSH project?	You abused git a bit by re-pushing the branches to a main repository
Do you have any suggestions for improvements?	See above
What is your overall impression of the workflow?	looks more complex than it really was, was quite easy to work with
Would you advice others to use the workflow?	Yes, with the above mentioned improvements

B.2 External Surveys

Please rate your experience and proficeny using the following technologies:

VCS (1-5)	3
DVCS (1-5)	3
Programming in general (1-5)	4
VCS experience	
Subversion	n
Git	n
CVS	n
Mercurial	y
BitKeeper	n
Bazaar	n
Other	
Do you have any previous experience with VCSs? If so, please describe.	
Current Project Version Control Use	
What version control software are you currently using?	Mercurial
How do you work with your VCS tools?	Pull,merge,commit,Push
Would you advice other to use your workflow?	y

Please rate your experience and proficeny using the following technologies:	
VCS (1-5)	4
DVCS (1-5)	4
Programming in general (1-5)	4
VCS experience	
Subversion	n
Git	n
CVS	n
Mercurial	n
BitKeeper	n
Bazaar	n
Other	
Do you have any previous experience with VCSs? If so, please describe.	
Current Project Version Control Use	
What version control software are you currently using?	Mercurial
How do you work with your VCS tools?	pull,merge,commit,push
Would you advice other to use your workflow?	y

Please rate your experience and proficeny using the following technologies:	
VCS (1-5)	3
DVCS (1-5)	4
Programming in general (1-5)	4
VCS experience	
Subversion	n
Git	n
CVS	n
Mercurial	n
BitKeeper	n
Bazaar	n
Other	
Do you have any previous experience with VCSs? If so, please describe.	
Current Project Version Control Use	
What version control software are you currently using?	Mercurial
How do you work with your VCS tools?	”Small”, atomic commits. Good communication to avoid merge conflitcs.
Would you advice other to use your workflow?	y

Please rate your experience and proficeny using the following technologies:

VCS (1-5)	3
DVCS (1-5)	2
Programming in general (1-5)	4
VCS experience	
Subversion	y
Git	y
CVS	n
Mercurial	n
BitKeeper	n
Bazaar	n
Other	
Do you have any previous experience with VCSs? If so, please describe.	
Current Project Version Control Use	
What version control software are you currently using?	SVN, Sharepoint
How do you work with your VCS tools?	
Would you advice other to use your workflow?	y

Please rate your experience and proficeny using the following technologies:

VCS (1-5)	3
DVCS (1-5)	?
Programming in general (1-5)	3
VCS experience	
Subversion	y
Git	n
CVS	n
Mercurial	n
BitKeeper	n
Bazaar	n
Other	
Do you have any previous experience with VCSs? If so, please describe.	
Current Project Version Control Use	
What version control software are you currently using?	Subversion
How do you work with your VCS tools?	As a plugin in eclipse
Would you advice other to use your workflow?	y

Please rate your experience and proficeny using the following technologies:

VCS (1-5)	4
DVCS (1-5)	4
Programming in general (1-5)	4
VCS experience	
Subversion	y
Git	y
CVS	n
Mercurial	n
BitKeeper	n
Bazaar	n
Other	
Do you have any previous experience with VCSs? If so, please describe.	
Current Project Version Control Use	
What version control software are you currently using?	SVN, Sharepoint for documents
How do you work with your VCS tools?	
Would you advice other to use your workflow?	n

Please rate your experience and proficeny using the following technologies:

VCS (1-5)	3
DVCS (1-5)	?
Programming in general (1-5)	3
VCS experience	
Subversion	y
Git	n
CVS	n
Mercurial	n
BitKeeper	n
Bazaar	n
Other	
Do you have any previous experience with VCSs? If so, please describe.	
Current Project Version Control Use	
What version control software are you currently using?	Subversion
How do you work with your VCS tools?	Dunno
Would you advice other to use your workflow?	

Please rate your experience and proficeny using the following technologies:

VCS (1-5)	3
DVCS (1-5)	3
Programming in general (1-5)	3
VCS experience	
Subversion	y
Git	n
CVS	n
Mercurial	n
BitKeeper	n
Bazaar	n
Other	
Do you have any previous experience with VCSs? If so, please describe.	
Current Project Version Control Use	
What version control software are you currently using?	Subversion
How do you work with your VCS tools?	
Would you advice other to use your workflow?	

Please rate your experience and proficeny using the following technologies:

VCS (1-5)	3
DVCS (1-5)	3,1
Programming in general (1-5)	3
VCS experience	
Subversion	y
Git	n
CVS	n
Mercurial	y
BitKeeper	n
Bazaar	n
Other	
Do you have any previous experience with VCSs? If so, please describe.	Gaming project
Current Project Version Control Use	
What version control software are you currently using?	Mercurial
How do you work with your VCS tools?	Pull,commit,push
Would you advice other to use your workflow?	

Please rate your experience and proficeny using the following technologies:

VCS (1-5)	2
DVCS (1-5)	2
Programming in general (1-5)	3
VCS experience	
Subversion	y
Git	y
CVS	n
Mercurial	n
BitKeeper	n
Bazaar	n
Other	
Do you have any previous experience with VCSs? If so, please describe.	
Current Project Version Control Use	
What version control software are you currently using?	Git at ”work!” and Subversion in School
How do you work with your VCS tools?	As soon as I have made a change(the smaller the better) i push my changes to the remote repository. I also pull as much as possible to stay updated
Would you advice other to use your workflow?	y

Please rate your experience and proficeny using the following technologies:

VCS (1-5)	2
DVCS (1-5)	3
Programming in general (1-5)	2
VCS experience	
Subversion	y
Git	y
CVS	n
Mercurial	n
BitKeeper	n
Bazaar	n
Other	
Do you have any previous experience with VCSs? If so, please describe.	
Current Project Version Control Use	
What version control software are you currently using?	We are using Git with git-bash and egit plugin for eclipse
How do you work with your VCS tools?	Push and pull oft so my local repository is in latest version if no errors exists.
Would you advice other to use your workflow?	y

Please rate your experience and proficeny using the following technologies:

VCS (1-5)	3
DVCS (1-5)	4
Programming in general (1-5)	3
VCS experience	
Subversion	y
Git	y
CVS	n
Mercurial	n
BitKeeper	n
Bazaar	n
Other	
Do you have any previous experience with VCSs? If so, please describe.	
Current Project Version Control Use	
What version control software are you currently using?	git
How do you work with your VCS tools?	kolla jenkins direkt på morgonen och han är blå så kör en pull annars fixa så han blir blå. därefter under dagen såfort som funktionalitet börjar kompilera som det ska så commit och kontinuerligt under dagen göra pull så ofta som möjligt så man inte kommer efter mergetåget. när saker och ting börjar likna nånting så kör push, var inte rädd för att bryta bygget då det alltid går att reseta.
Would you advice other to use your workflow?	y

C Build Metadata

ID	Date	Status	Duration
2	2011-03-08T09:22:49	SUCCESS	3.221
3	2011-03-08T09:56:12	SUCCESS	2.444
4	2011-03-08T10:48:24	SUCCESS	2.363
5	2011-03-08T10:52:26	SUCCESS	2.409
6	2011-03-08T11:23:50	SUCCESS	2.311
7	2011-03-08T12:42:08	SUCCESS	2.391
8	2011-03-08T15:46:21	SUCCESS	2.405
9	2011-03-08T15:52:20	SUCCESS	2.369
10	2011-03-08T16:35:25	SUCCESS	2.404
11	2011-03-08T16:38:48	SUCCESS	2.309
12	2011-03-08T16:39:23	SUCCESS	3.354
13	2011-03-09T11:34:21	SUCCESS	5.047
14	2011-03-09T15:22:29	SUCCESS	2.505
15	2011-03-10T08:38:57	SUCCESS	5.047
16	2011-03-10T09:12:15	SUCCESS	2.729
17	2011-03-10T09:56:34	SUCCESS	2.629
18	2011-03-10T10:18:54	FAILURE	1.334
19	2011-03-10T10:21:59	FAILURE	0.267
20	2011-03-10T10:23:55	SUCCESS	2.917
21	2011-03-10T10:26:10	SUCCESS	2.942
22	2011-03-10T10:57:14	SUCCESS	2.903
23	2011-03-10T10:59:44	FAILURE	0.288
24	2011-03-10T11:08:43	FAILURE	0.394
25	2011-03-10T11:13:39	FAILURE	0.066
26	2011-03-10T11:21:07	SUCCESS	5.578
27	2011-03-10T15:19:26	SUCCESS	4.309
28	2011-03-11T09:28:28	SUCCESS	6.252
29	2011-03-11T10:32:45	FAILURE	2.225
30	2011-03-11T11:21:10	SUCCESS	3.933
31	2011-03-11T14:19:55	SUCCESS	4.089
32	2011-03-11T14:40:33	SUCCESS	4.115
33	2011-03-11T15:28:41	SUCCESS	3.971
34	2011-03-11T17:26:00	FAILURE	2.19
35	2011-03-11T17:30:38	SUCCESS	4.869
36	2011-03-14T10:30:56	SUCCESS	5.669
37	2011-03-14T10:35:08	FAILURE	2.437
38	2011-03-14T11:00:03	SUCCESS	4.883
39	2011-03-14T13:59:10	SUCCESS	4.891
40	2011-03-14T14:02:41	SUCCESS	5.153
41	2011-03-14T14:55:48	SUCCESS	5.102
42	2011-03-14T15:58:47	SUCCESS	5.023
43	2011-03-14T17:17:53	SUCCESS	4.895
44	2011-03-15T10:05:29	SUCCESS	5.359
45	2011-03-15T10:06:29	SUCCESS	4.884
46	2011-03-15T13:04:07	SUCCESS	5.062
47	2011-03-15T13:30:42	SUCCESS	5.123
48	2011-03-15T14:29:41	SUCCESS	4.908
49	2011-03-15T14:49:19	SUCCESS	5.296
50	2011-03-16T11:50:37	SUCCESS	14.149

ID	Date	Status	Duration
51	2011-03-16T14:47:55	SUCCESS	11.721
52	2011-03-16T16:24:41	SUCCESS	12.463
53	2011-03-16T17:14:58	SUCCESS	11.973
54	2011-03-17T12:38:50	SUCCESS	13.45
55	2011-03-18T15:30:00	SUCCESS	14.336
56	2011-03-21T09:20:18	SUCCESS	13.301
57	2011-03-21T14:18:45	SUCCESS	11.934
58	2011-03-21T14:19:56	SUCCESS	11.621
59	2011-03-21T15:38:32	SUCCESS	11.756
60	2011-03-21T17:02:26	SUCCESS	11.818
61	2011-03-22T09:15:03	SUCCESS	12.539
62	2011-03-22T11:03:16	SUCCESS	12.705
63	2011-03-22T12:23:08	SUCCESS	11.452
64	2011-03-22T14:40:03	SUCCESS	12.941
65	2011-03-22T14:45:30	SUCCESS	12.648
66	2011-03-22T17:15:12	SUCCESS	13.191
67	2011-03-24T13:06:43	SUCCESS	6.834
68	2011-03-24T13:09:50	SUCCESS	5.22
69	2011-03-25T08:20:14	SUCCESS	8.63
70	2011-03-25T09:07:21	SUCCESS	5.449
71	2011-03-25T14:07:43	SUCCESS	5.928
72	2011-03-25T14:09:03	SUCCESS	5.809
73	2011-03-25T14:09:51	SUCCESS	6.024
74	2011-03-28T15:31:41	SUCCESS	7.12
75	2011-03-28T15:36:19	SUCCESS	5.794
76	2011-03-29T11:04:18	SUCCESS	8.805
77	2011-03-29T13:04:08	SUCCESS	6.648
78	2011-03-29T13:57:07	SUCCESS	6.468
79	2011-03-29T17:05:58	SUCCESS	6.721
80	2011-03-30T10:50:57	FAILURE	10.788
81	2011-03-30T10:55:25	FAILURE	8.971
82	2011-03-30T10:57:01	FAILURE	9.014
83	2011-03-30T10:58:16	FAILURE	8.98
84	2011-03-30T10:59:03	FAILURE	9.048
85	2011-03-30T10:59:56	FAILURE	9.914
86	2011-03-30T11:05:25	FAILURE	9.315
87	2011-03-30T11:21:26	FAILURE	8.99
88	2011-03-30T11:26:57	FAILURE	8.949
89	2011-03-30T11:29:36	FAILURE	8.451
90	2011-03-30T12:45:04	FAILURE	8.368
91	2011-03-30T12:45:43	SUCCESS	11.057
92	2011-03-30T14:26:04	SUCCESS	11.262
93	2011-03-31T10:35:36	SUCCESS	12.099
94	2011-03-31T12:24:31	SUCCESS	12.289
95	2011-03-31T14:42:52	SUCCESS	11.327
96	2011-04-01T11:11:04	SUCCESS	13.41
97	2011-04-01T11:19:07	SUCCESS	11.66
98	2011-04-01T11:24:32	SUCCESS	11.881

ID	Date	Status	Duration
99	2011-04-01T12:06:25	SUCCESS	11.952
100	2011-04-01T13:15:29	SUCCESS	11.475
101	2011-04-01T15:08:00	SUCCESS	11.645
102	2011-04-01T15:21:29	SUCCESS	11.579
103	2011-04-05T08:45:34	FAILURE	15.995
104	2011-04-05T08:55:06	SUCCESS	11.716
105	2011-04-05T09:11:45	SUCCESS	15.541
106	2011-04-05T10:47:05	SUCCESS	10.368
107	2011-04-05T11:43:08	SUCCESS	11.021
108	2011-04-05T11:47:50	SUCCESS	10.931
109	2011-04-05T12:15:24	SUCCESS	11.147
110	2011-04-05T12:28:28	SUCCESS	11.22
111	2011-04-05T13:28:52	SUCCESS	10.807
112	2011-04-05T14:14:02	SUCCESS	11.331
113	2011-04-11T08:52:29	SUCCESS	16.533
114	2011-04-11T09:37:21	SUCCESS	11.818
115	2011-04-12T08:40:28	SUCCESS	16.813
116	2011-04-12T12:44:33	SUCCESS	12.365
117	2011-04-12T15:51:42	SUCCESS	12.317
118	2011-04-13T10:56:24	SUCCESS	13.602
119	2011-04-13T11:00:18	SUCCESS	11.536
120	2011-04-13T11:10:17	SUCCESS	11.585
121	2011-04-13T12:16:34	SUCCESS	11.53
122	2011-04-13T13:51:59	SUCCESS	11.253
123	2011-04-13T13:53:32	SUCCESS	11.168
124	2011-04-13T14:30:41	SUCCESS	11.241
125	2011-04-13T14:48:10	SUCCESS	11.564
126	2011-04-13T16:29:21	SUCCESS	11.311
127	2011-04-14T10:26:32	SUCCESS	13.992
128	2011-04-15T09:19:02	SUCCESS	14.117
129	2011-04-15T15:45:58	SUCCESS	12.67
130	2011-04-15T15:52:08	SUCCESS	11.522
131	2011-04-18T08:16:43	SUCCESS	15.56
132	2011-04-18T08:27:16	SUCCESS	11.681
133	2011-04-18T08:28:08	SUCCESS	11.787
134	2011-04-18T13:22:33	SUCCESS	12.845
135	2011-04-18T13:57:01	SUCCESS	11.518
136	2011-04-18T14:57:48	SUCCESS	12.223
137	2011-04-19T10:44:49	SUCCESS	13.898
138	2011-04-20T13:38:23	SUCCESS	14.164
139	2011-04-26T10:25:04	SUCCESS	13.77
140	2011-04-29T07:44:47	SUCCESS	23.487
141	2011-05-02T12:58:33	SUCCESS	16.739
142	2011-05-03T08:42:25	SUCCESS	16.387
143	2011-05-03T16:15:50	SUCCESS	15.738
144	2011-05-03T16:39:32	SUCCESS	13.588
145	2011-05-03T19:35:52	SUCCESS	13.722
146	2011-05-04T08:59:27	SUCCESS	16.842
147	2011-05-04T14:32:24	SUCCESS	13.729
148	2011-05-05T08:01:22	SUCCESS	19.343
149	2011-05-05T10:06:31	SUCCESS	15.222
150	2011-05-05T14:22:38	SUCCESS	16.894
151	2011-05-05T14:24:49	SUCCESS	13.448
152	2011-05-05T14:36:59	SUCCESS	13.566
153	2011-05-16T13:44:32	SUCCESS	19.865