CSCC01: Reading Summary
Kalindu De Costa 1001392619

# Why are Software Projects moving from Centralized to Decentralized Version Control Systems?

## 1. Summary

This paper summarizes the differences of Decentralized Version Control Systems (DVCS) with the Centralized Version Control Systems (CVCS) and provides some legitimate examples of projects switching from CVCS to DVCS and their rationale and perceived benefits to justify the transition. DVCS have peaked the interest of many projects as it addresses many issues of CVCS like, difficulty in repeated merges of branches. However, this paper suggests that DVCS may also introduce new issues that are yet unknown.

Unlike a CVCS a DVCS relax the requirement of a central master repository. With a DVCS each checkout is itself a first-class repository, a full copy of the master repository which includes the complete commit history. This eliminates the issue of write-access and every developer has a personal repository regardless of whether they are an accepted committer to the project. Moreover, DVCS allows for easy branching and easy merging of branches compared to those of CVCS. This encourages better projects managements and encourages a practice called feature branches (de Alwis & Sillito, 2009). Further, DVCS allows for disconnected development. This allows for reliable software to cater to teams developing across distributed locations. Finally, DVCS naturally leads to source code repository being replicated to multiple locations which reduces the risk of a disaster scenario like, source code corruption and loss of source code.

Projects like Perl, OpenOffice, NetBSD and Python are currently in the process of or have already undergone the transition form a CVCS to a DVCS. The transitioning process requires a large amount of work and one of the discouraging reasons of transitioning is the required amount of effort and manpower. However, for projects like NetBSD, whom have maintained its source code in CVCS repository since 1993 (de Alwis & Sillito, 2009), the transition to a DVCS is very beneficial for its easy branching and merges.

## 2. Findings and Benefits

The main reason for the transition of CVCS to DVCS is to improve support for non-committers and provide first-class access to all developers. According to Python developers, this was the main reason of transition to a DVCS since contributors cannot benefit from CVCS for their work. Similarly, Perl put emphasis on open-source DVCS to ensure that all tools are available to all members of the community.

DVCS supports atomic changes. NetBSD and OpenOffice projects, most required this feature due to previous repository corruptions that have occurred when carrying out repository-wide atomic commits. In some cases, these corruptions can go undetected.

As explained previously, DVCS allows for easy branch merging. This also relates to simple automatic merging and repeated merges, specifically for long-lived branches. The python project found this very important to encourage their developers to keep their branches up-to-date and reduce risk of their branch becoming stale. Moreover, the Perl project found this feature useful to synchronize branches before delivering a patch to prevent version mismatches.

Finally, DVCS contains improved support for experimental changes and support for disconnected operations. With cheap branches, projects can deliver experimental features to non-committers to undergo testing and acquire feedback.

## 2.1. Challenges

The work required to transition form a CVCS to a DVCS is significant, and projects that do switch has compelling reasons to this decision. One of the challenges that projects face during this transition is the change of the teams' development process. The OpenOffice team for example looked for easy integration into their current development process and the Python project heavily documented the changes required to their development process once the transition has been made.

Loss of metadata from their previous VCS was another concern of teams like NetBSD and Perl developer. This metadata, such as version numbers and embedded references from commits are too relevant to ignore and must be transferred when transitioning to a DVCS.

Many developers of NetBSD showed concern for human-identifiable commit identifiers like incrementing version numbers, one of which SHA-1 based commit identifiers in DVCS cannot satisfy. This shows problematic behavior when considering tags and references within emails and work items. So, this change requires another level of adaptation for the developers and their long lasting non-committers.

Finally, DVCS requires a significant change in a developer's routine repository management and can slow down the development process for large period of time. Many project teams will need to invest a large amount of time to retrain their developers to effectively take advantage of this new system.

## 3. Thoughts

I feel that DVCS are far superior to CVCS. The fact of each developer containing a firs-class copy of a repository is a huge advantage to avoid corruption and disasters. Moreover, it allows for easy access to the complete project and team contributions which can result in better code management and defect-less source code. Further, DVCS allows for better security for the repository due to the lack of a master repository. Finally, the easy branching and merges of DVCS can greatly improve the efficiency of managing large projects and improve the development process for all developers.

## References

1. de Alwis, B., & Sillito, J. (2009). Why are software projects moving from centralized to decentralized version control systems?. *2009 ICSE Workshop On Cooperative And Human Aspects On Software Engineering*. doi: 10.1109/chase.2009.5071408