Paper1

Title: Version Control System: A Review

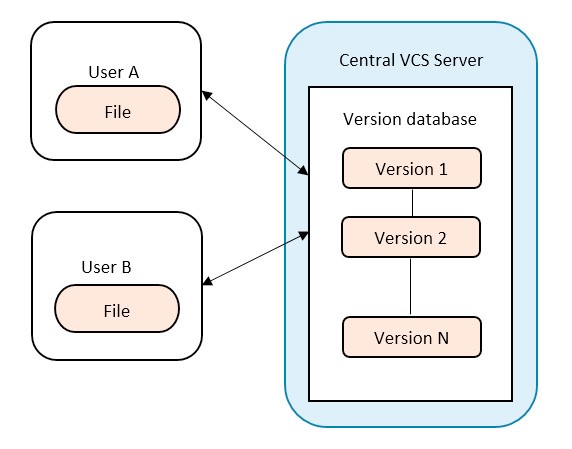
Authors: Nazatul Nurlisa Zolkifli, Amir Ngah\*, Aziz Deraman

Published in :

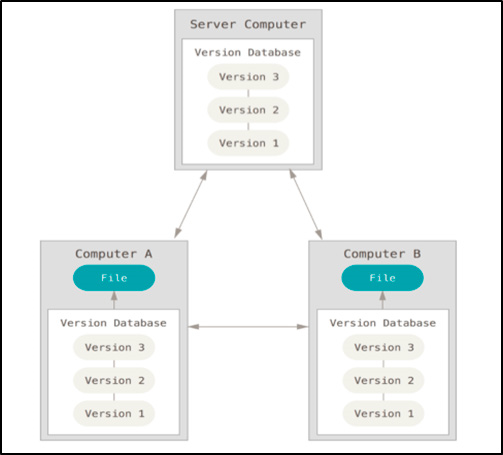
Introduction to VCS:

* Version Control System (VCS) is a system that manages the development of an evolving object.
* VCS is also known as Revision Control System

Centralised VCS:



Distributed VCS:



Comparison between Centralised and Distributed

Table 1. Comparison between CVCS and DVCS.

|  |  |  |
| --- | --- | --- |
| Version Control System | CVCS | DVCS |
| Repository | There is only one central repository which is the server. | Every user has a complete repository which is called local repository on their local computer. |
| Repository Access | Every user who needs to access the repository must be connected via network. | DVCS allows every user to work completely offline. But user need a network to share their repositories with other users. |
| Example of VCS Tools | Subversion, Perforce Revision  Control System | Git, Mercurial, Bazaar, BitKeeper |
| Software Characteristics that suitable | 1. Projects that allow only several users to contribute to the software development. 2. Team located in a single site. | 1. DVCS is suitable for a single or more developers because the project repository is distributed to all the developers and this ability offer a great improvement for the projects. 2. It also can be applied for a small or big software projects because it makes less difficult for normal users to contribute to the development. 3. Team located in multiple site or different countries and different timezones. |

Applications:

* Software merging
* Collaboration Modelling
* Software changes
* Software branching
* Open Source Software Projects

Conclusion:

Software developers should have a rudimental understanding of what VCS is and which type of VCS suits them. The adoption of a VCS is a must in software development. It helps software developers manage their codes easily because it is common to have a lot of changes involving addition or deletion of features. In order to adopt a VCS, a software developer must know and perfectly understand which approach should be used as it will affect the whole project and team. It is also important for them to have the knowledge of different approaches of VCS because the various approaches will affect their software development process differently.

Paper:2

Title:How Do Centralized and Distributed Version Control Systems Impact Software Changes?

Authors: Caius Brindescu, Mihai Codoban, Sergii Shmarkatiuk, Danny Dig

**Table 1: Demographics of survey respondents**

1. **Programming experience ( years )**

*<* 2 2 - 5 5 - 10 10 - 15 15 - 20 *>* 20

1.83% 11.10% 30.49% 30.61% 13.90% 12.07 %

1. **Project type**

|  |  |  |  |
| --- | --- | --- | --- |
| Proprietary Open source Research software software project | | Personal project | Other |
| 85.09% 6.97% 4.64% | | 3.06% | 0.24 % |
|  | **(c) Team size** |  |  |
| 1 2 – 5 | 6 - 10 11 - 25 | 26 - 100 | *>*100 |
| 5.87% 42.30% | 23.72% 15.65% | 8.19% | 4.28% |

**d**

**(**

**)**

**Project**

**age**

*<*

6

mo

6

mo

-

1

yr

1

yr

-

2

yrs

*>*

2

yrs

%

21.27

%

46.82

%

13.33

%

18.58

**VCS**

**used**

**predominantly**

**(**

**e**

**)**

Microsoft

Git SVN Hg CVS Other TFS

52.68% 20.37% 12.07% 8.54% 1.10% 5.24 %

**Table 2: Repository corpus.**

|  |  |  |  |
| --- | --- | --- | --- |
| Type. Repositories | Commits | Authors | Total  LOC  changed |
| SVN 52 | 95571 | 451 | 270 M |
| Hybrid 29 | 151004 | 2249 | 89 M |
| Git 51 | 111725 | 3190 | 50 M |
| Total 132 | 358300 | 5890 | 409M |

# RESULTS

**Observation 1:** DVCS repositories have a smaller commit size than CVCS repositories, in terms of lines of

**Observation 2:** 76% of the developers split their com

mits. The percentage is higher for distributed version control systems (81.25%), compared to centralized ones (67.89%).

**Observation 3:** Overall, developers choose to split their commits using the intent of change.

**Observation 11:** Most projects use an Issue Tracking System

**Observation 17:** Commit size tends to become smaller as projects get older

**Observation 13:** Large teams squash commits more often

# CONCLUSIONS

In this paper we present the first in-depth study to measure the impact of DVCS on software change. To this end we ran a survey with 820 participants and analyzed a corpus of 132 repositories.

We found that the use of CVCS and DVCS have observable effects on developers, teams and processes. The most surprising findings are that (i) the size of commits in DVCS was smaller than in CVCS, (ii) developers split commits (group changes by intent) more often in DVCS, and ( iii ) DVCS commits are more likely to reference issue tracking labels. These show that DVCS contain higher quality commits compared to CVCS due to their smaller size, cohesive changes and the presence of issue tracking labels.

Paper:3

Title: Software Development and Collaboration: Version Control Systems and Other Approaches

Authors:

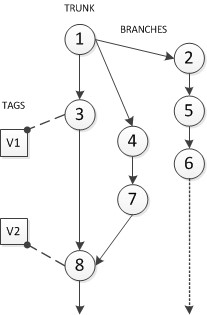


Fig. 1. VCS Repository structure

Inside a repository, the current working versions of the source code are stored within the *Trunk* of the history tree. This *Trunk* represents the main development effort. It is possible to see *Trunk* referred with similar terms such as *Baseline* or *Mainline*. In a certain point in time, a version could be branched, creating a new *Branch* to the tree. A branch can be developed parallel to the main version and then be merged to the trunk again or follow a completely different evolution. In some systems, a branch can be also declared as a *Mainline*. A *Revision* is a version of a file. The other main concept behind a VCS tree is *Tag* or *Label*, that identifies a concrete set of files in the timeline with a human readable and comprehensive name such as a version number for a main release or a version name.

When working with a repository, several actions can be performed. The first step to create a new repository requires to make an *Import*, the action of copying a local directory to the repository for the first time which implies that from that point, a version control will be associated to this directory. In the case of working with a previously existing repository, the first step requires to make a *(*Checkout), creating a local copy of the current revision of the repository and allowing the user to work with it. This checkout action is only needed the first time because once we have a local copy, making an *Update* over a file or a group of them, changes made to the file(s) by other users are merged in our copy. The action of *Commit* or *Checkin* is to write changes made in our local copy to the repository. A very important action in any kind of VCS is the *Merge* action, allowing to merge different branches, merge local copies with the working copies on the repository (or the way around) among others. Although *Export* action is very similar to a checkout, the difference between them is that a exported version does not keep any version history. This action is useful for example for obtaining release versions ready to be distributed.

*A. CVS*

Concurrent Versions System, also known as *CVS*, is an open source software for version control.

it is the most famous exponent of centralized VCSs

famous exponent of centralized VCSs. Distributed under a GNU license[8], it was first designed and coded by Brian

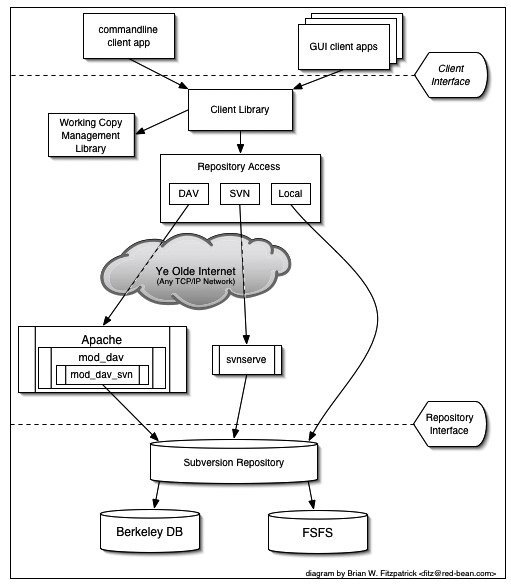
When a user successfully makes a check in, the version number of all involved files are automatically increased and a description given by the user, the author’s name and the current date are written into a log file. If a user working in a file in his local copy receive a new update made by other user to the same file, CVS try to automatically merge both versions into one but if it is not possible and there exists a conflict, it is up to the user to decide how to proceed to solve the problem, CVS just mark the conflicting parts of the source code.

Within a CVS repository, different projects stored are referred as *modules*. Inside those modules, source files of projects are stored using a delta compression algorithm. CVS also allows anonymous read access, also known as *Anonymous CVS*

Shortcomings of CVS.

* CVS does not version the moving or renaming of files and directories.
* Limited support for Unicode and non-ASCII filenames.
* A very important issue of CVS is that commits are not atomic
* Limited tagging operations

*B. Subversion*



Different from CVS, Subversion has a native support for client-server operation and allows an advance network layer[12] through the use of WebDAV module from Apache 2. Because of this advanced network layer, there are several ways to operate with a repository. The file system can be accessed directly by the users at a local level through an URL scheme (*“file://path”*) but also it can be accessed (thanks to the DAV module) over the http(s) protocol. In addition to this access method, svn defines its own protocol over TCP/IP that also allows ssh tunneling through a “svn+ssh://host/path” access scheme.

3.Git

DVCS allows every user to work completely offline. But user need a network to share their repositories with other users.

One of the most important differences between Git and almost any other VCS (besides all that implies being a DVCS), is the way how Git treat data. In Git, instead of keeping a reference to the changes made to files, every time a user commit a new version, Git takes an snapshot of the state of all files and copy them as a new version. As a direct consequence of this data handling, Git becomes a mini filesystem itself, allowing to develop very useful and powerful tools in top of it. Besides the advantages of working with a local repository (e.g., offline working, fast operations, etc.), other important improvement introduced by Git is integrity. Everything under Git control has a checksum, allowing to detect any kind of changes on files and preventing corruption. When a user modifies a file in his working directory, it gets the status of modified and then the user needs to stage the file. When a file is staged, it means that the file is ready to be committed. Once a file is committed, it is safely stored in the repository.

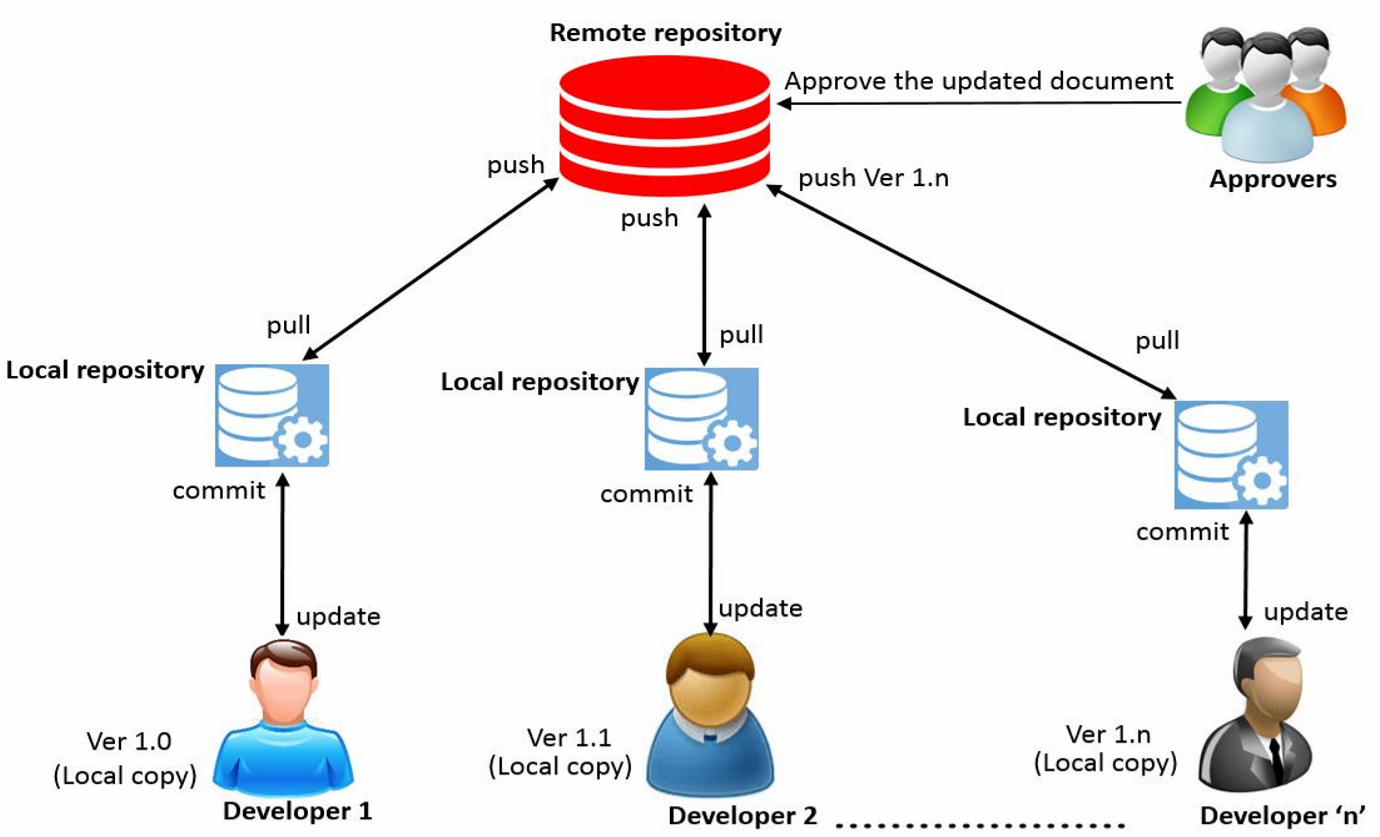
Conclusion:

* In the field of VCSs, we can identify two main groups of systems: centralized (CVCSs) and decentralized or distributed (DVCSs).
* There exist a lot of implementations of both systems architectures. Among them, CVS and SVN are most used CVCSs, being the second one a natural evolution of the other, improving several lacks of features and operation problems.
* Even though SVN is right now the most used VCS, it seems that gradually more and more important projects are migrating their version control systems toward a distributed solution such as Git.

Paper:3

Title: Decentralized Document Version Control using Ethereum Blockchain and IPFS

Author: N. Nizamuddin1 K. Salah1 M. Ajmal Azad2 J. Arshad3 M. H. Rehman4



**Fig. 1.** Traditional document version control systems

Main work:

* a blockchain-based solution for document version control for digital documents using Ethereum smart contracts.

**Fig**

**.**

**2**

**. Applications and Objectives of Blockchain based version control systems**



Blockchain based Version

Control Systems



Applications



Land Record Management

[14]



Healthcare [19], [20]



Objectives

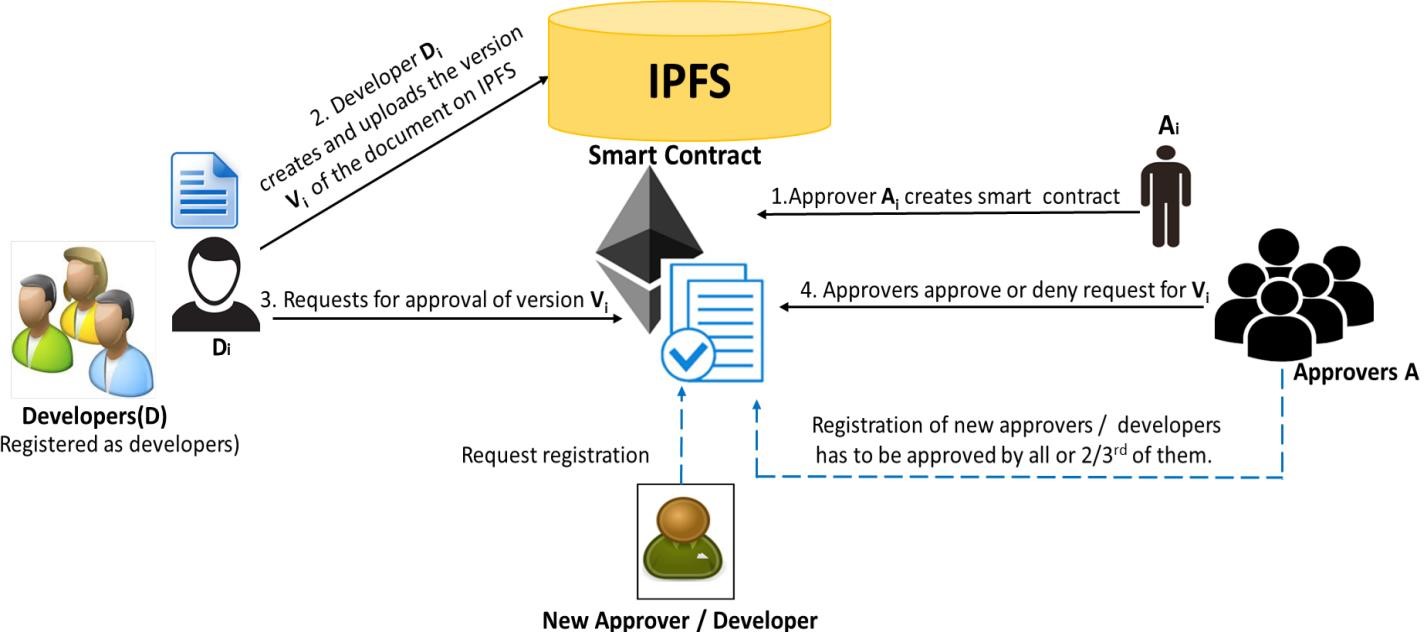


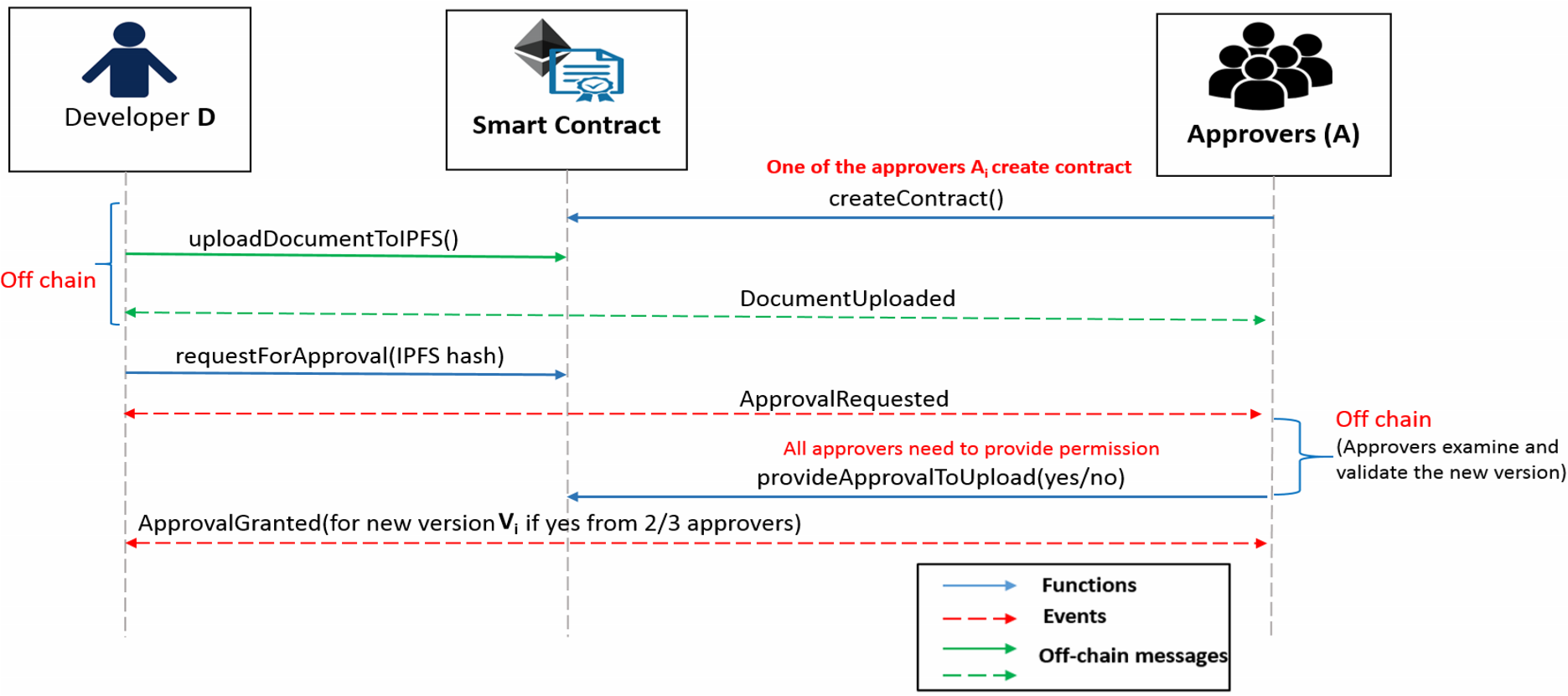
Secure Storage [15,17]



Data Integrity [16]

**PROPOSED BLOCKCHAIN-BASED SOLUTION**





**Table 2**: Comparison between conventional and blockchain based version control frameworks.

|  |  |  |  |
| --- | --- | --- | --- |
| **Features** | **Centralized (Conventional)** | **Decentralized (Conventional)** | **Blockchain based (Decentralized)** |
| Mode of Operation | * Client-server * Cloud-based * P2P | * Client-server * Cloud-based * P2P |  P2P |
| Point(s) of Failure |  Single |  Multiple |  None |
| Merge Strategy |  Linear  Random |  Linear  Random |  Consensus Based |
| Trust Level |  Low |  Medium |  High |
| Graphical User Interfaces | * Web-based * Desktop-based * Integrated | * Web-based * Desktop-based * Integrated | * Web-based * Integrated |
| Monopoly |  High |  High |  Low |
| Strength |  Ad hoc merge and  split |  Ad hoc merge and  split |  Consensus-based  merge and split |
| Weaknesses |  Roll back possible |  Roll back possible |  No Rollback |

**Conclusion:**

The proposed system is completely decentralized, secure, and resilient, eliminating dependency on the trusted third party