Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 135 (2018) 408–415

Procedia Computer Science

www.elsevier.com/locate/procedia

3rd International Conference on Computer Science and Computational Intelligence 2018

Version Control System: A Review

Nazatul Nurlisa Zolkifli, Amir Ngah*, Aziz Deraman

School of Informatics and Applied Mathematics, Universiti Malaysia Terengganu, 21030 Kuala Nerus, Terengganu, Malaysia

Abstract

Version Control Systems (VCS) have been used by many software developers during project developments as it helps them to manage the source codes and enables them to keep every version of the project they have worked on. It is the way towards managing, organizing, and coordinating the development of objects. In Software Engineering, software developers need to collaborate with each other to develop a better project. Thus, VCS is very useful because it also supports a collaborative framework that makes it easy for software developers to work together effectively. Without VCS, collaboration is very challenging. This paper discusses the background and the related works about VCS that have been studied by researchers. The purpose of this paper is to convey the knowledge and ideas that have been established on VCS.

Keywords: Version Control Systems; Distributed Version Control Systems; Centralized Version Control Systems; Software Development; Collaborative Development

1. Introduction

Version Control System (VCS) is a system that manages the development of an evolving object. In other words, it is a system that records any changes made by the software developers. There are a lot of uses for VCS in software development that makes the development process easier and faster. VCS is also known as Revision Control System

* Corresponding author. Tel.: +6096683463
E-mail address: amirnma@umt.edu.my

(RCS), Software Configuration Management, Source Code Management (SCM), and Source Code Control [1]. Despite the various terms, they literally mean the same except that VCS focuses more on the versioning aspects.

In the software development process, it is normal for software developers to continually make changes in pieces of codes and other files that involve addition and deletion of a feature. It is realized that several revisions will be made before producing the final version. It is difficult to manage and organize the codes and the files because the number of revisions grows larger due to larger and complex systems. Hence, the existence of the VCS really helps software developers to accelerate and simplify the development process.

Without VCS, the software developers are enticed to keep various duplicates of code on their computer. This is risky because it is easy to change or erase a document or file in the wrong copy of code, possibly leading to losing work. Version control system will take care of this issue by managing all versions of the code. The adoption of VCS is progressively mandated to empower all the software developers who work on the same project to work together effectively towards project milestones [2]. It is an approach of managing the tasks of multiple team members who are sometimes at different locations.

The remainder of this paper is organized as follows: Section 2 discusses the basic concept of VCS. Subsequently, Section 3 will discuss the related works that have been studied by researchers about VCS. This paper ends with the advantages of using VCS in Section 4 and the conclusion is presented in Section 5.

2. Version Control System

According to Otte, there are two different approaches to VCSs, which are the Centralized Version Control System (CVCS) and the Distributed Version Control System (DVCS) [1]. CVCS is a centralized model with one central repository while DVCS is a distributed model without a central repository but has a local repository for every user. The most commonly used CVCS tools are CVS and Subversion [2,3]. Since the emergence of DVCS tools such as Git, Mercurial, Bazaar, and BitKeeper, many open and closed source projects have been proposing to move or have already moved their source code repositories to a DVCS [3]. Summary of the comparison between CVCS and DVCS is shown in the Table 1 below.

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	i. Projects that allow only several users to contribute to the software development. ii. Team located in a single site.	i. 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. ii. 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. iii. Team located in multiple site or different countries and different timezones.

Normally, software development with a VCS will begin with a new project. Software developers are also allowed to import existing projects into the VCS. Thereafter, software developers need to check out a version of the project into their working directory in order to work on the project and make changes to the codes. Once they are satisfied with the changes they have made, they can commit the changes to the repository with an explanatory message that describes the changes. This is followed by the next step, whereby the software developers need to synchronize their private version with the changes of the other team members who have committed, with the purpose of getting the new changes made by the others. Lastly, the release's name will be tagged or labelled to roll out a release version.

2.1. Centralized Version Control Systems (CVCS)

Similarly as with several other programming bundles or ideas, as the prerequisites continue developing, clients feel that the local version control systems constrain their activities [11]. People are not ready to work cooperatively on a similar project, as the records with their versions are put away in some person's nearby personal computer and are not open to other individuals who work on similar documents. CVCS is an approach that enable the developers to work cooperatively. It stores a master copy of files history and in order to read, retrieved, commit new changes to a certain versions, they need to contact a server [22].

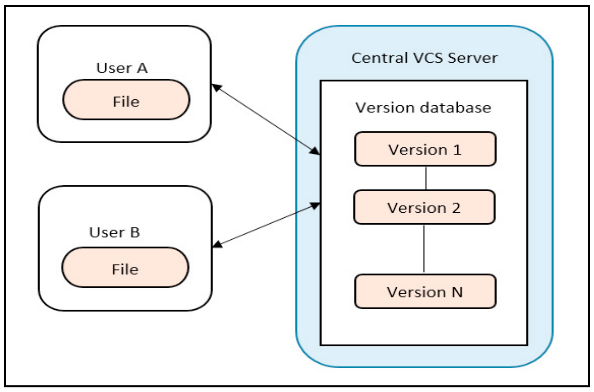

Fig. 1 Somasindaram R. (2013). Centralized version control [Digital image]. Retrieved April, 2018.

As shown as Fig. 1, CVCS have a single repository which is basically the server[2,11]. All the file were kept in the server that everybody has access to from their local computer. All the developers make changes against this repository through a checkout taken from the repository but only the last version of the files are retrieved. As the files are stored in the server, it means that any changes made will automatically shared with other developers.

The developers otherwise known as the contributors and committers, must be the core developers who perform basic tasks such as creating branches, merging branches or reverting changes to previous states [10]. In spite of that, this restriction will affect the participation and authorship of new contributors because in order to perform basic tasks, they need to follow some steps in joining the process to become a core developer. Below are the processes that must be followed by the contributor to become a core developer who has the writing permission:

- A software developer will start as a passive user which means his/her activity cannot be traced. At this stage, the developer cannot submit any email message or bug reports as he/she is only allowed to visit the project's website and read the mailing list;
- From a passive user, the developers will become advanced users if they become familiar with the project. At this stage, the developers are allowed to send some messages to the mailing list. This activity can be tracked;
- Developers who have partial knowledge about the project may do some report bugs, submit comments, and make some modifications to the project;

- At this stage, any contributor who gives meaningful and important contributions regarding the project will be given an account with write access to code repository of the project. Developers who have this writing permission can commit any changes made; and
- The last process in order to be in a core group is by collaborating in the project with high level activity over a certain amount of time.

There is potential risk involved in using a CVCS because the users only have the last version of the files in their system for working purposes. Most commonly used CVCS are Concurrent Version System (CVS), Subversion (SVN), and Perforce.

2.2. Distributed Version Control Systems (DVCS)

Nowadays, Open Source Software (OSS) projects largely adopt the DVCS [12]. This is because of the risk of using the Local Version Control and CVCS. When a project is using a CVCS, it means that the history related to the project will be kept in a single place, while developers will lose the power to work collaboratively when using the local version control. If a developers are using CVCS, it will works well if they can contact the server but they cannot contact the server it will be a big problem for them [22]. The risks stated are the reasons why most OSS projects are adopting the new version control system for recording, managing, and propagating their source code changes. DVCS is a combination of advantages of both concepts and a hybrid system [11].

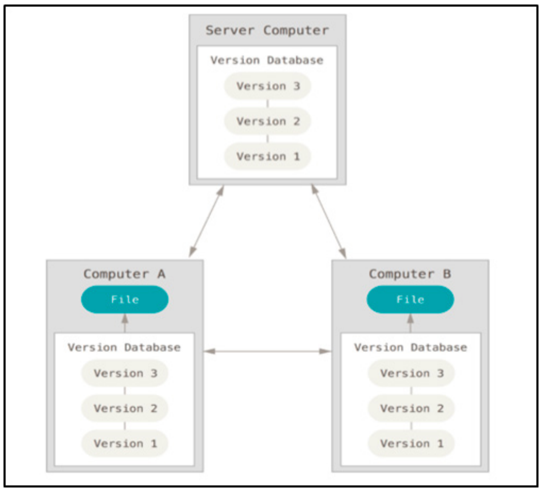

Fig. 2 Chacon S & Strauv B (2013). Distributed version control system [Digital image]. Retrieved April, 2018.

As shown in Fig. 2, fundamentally, DVCS is designed to act in both ways as it stores the entire history of the files on each machine locally[21] and it can also sync the local changes made by the user back to the server whenever required, so that the changes can be shared with the whole team [11]. Clients can communicate with each other and maintain their own local branches without having to go through a central server or repository. Then, synchronization takes place between the peers who decide which change sets to exchange. Clients do not just look into the latest snapshot of the files but they are able to totally copy the repository. Therefore, if any server breaks down, and these systems were collaborating using it, any of the client repositories can be mirrored back to the server to recover it. Every clone is genuinely a full backup of all the data. Examples of DVCS tools are Git, Bazaar, Mercurial, Bitkeeper, and Darcs [12, 23]. Bitkeeper and Bazaar are two of the earliest DVCS tools [19]. Git and Mercurial are the newest DVCS tools that improve the merging and branching features.

3. Application of VCS

Numerous researches have focused on version control systems. Subsequently, we grouped the related works that have been studied by researchers according to the application of VCS based on our analysis and discussed in the following sub-topics.

3.1. Software merging

Cavalcanti et al. discusses the comparison between the unstructured and semi-structured approaches over the Git, which focuses on the study of the activities of integration and conflict resolution in software projects [4]. As the authors pointed out, the conflict resolution strategies adopted by version control systems can be classified into three which are unstructured, structured, and semi-structured. Semi-structured approach is the combination of the strength of both, the expressiveness of the structured merge, and the generality of the unstructured merge [4]. The purpose of the semi-structured approach is to be responsible for providing the information regarding the software artifacts so that the conflicts that occur can be resolved automatically. 154 of merge scenarios from 10 software projects that used Git as the VCS were used to conduct the study. It was found that the semi-structured approach could extensively reduce the number of conflicts that occurred and it was able to resolve the ordering conflicts that rely on the software's elements order.

Cavalcanti discusses the empirical studies that have been conducted in order to identify false positive or spurious conflicts [13]. According to the writer, unstructured merge tools are entirely text-based so the tools will revolve the merge conflicts via textual similarity [13]. In contrast, the structured merge tools which are based more towards programming languages, will use the programming syntax to resolve the merge conflicts [14]. Semi-structured merge tools combine the structured and unstructured and will tackle the merge conflicts by exploiting the syntactic structure and semantics of the involved artifacts partially. The purpose of the empirical studies is to investigate whether the integration effort reduction (Productivity) without negative impact on the correctness of the merging process (Quality) is driven by the semi-structured and structured merge reduction on the number of reported conflicts in relation to the unstructured merge.

3.2. Collaboration Modelling

Alam et al. provided a thorough analysis in an article entitled *Towards Collaborative Modelling Using a Concern-Driven Version Control System* [16]. In the paper, the authors stated that the Concern-Oriented Reuse (CORE) approach is the idea for a collaboration in modelling. According to the authors, CORE is a novel reuse paradigm that assists broad-based model reuse. Software developers widely use VCS in order to team up and track development activities. The reuse in software development can be categorized by class libraries, services, and components facilitated by collaboration and coordination during the development activities [16]. As a result of the effectiveness of the reuse in software development, software developers can reuse the artifacts developed by other developers. The advancement in reuse in VCS allows software developers to be more collaborative. This approach will make collaboration in modelling easier, simpler, and faster with modular ways to settle conflicts that occur.

In addition, Debreceni et al. introduced *the MONDO COLLABORATION FRAMEWORK: Secure Collaborative Modelling over Existing System* [17]. The authors explained that the MONDO COLLABORATION FRAMEWORK provides a rule-based fine–grained model-level secure access control, property-based locking, and automated model merge integrated over existing VCS such as Subversion and Git for storage and version control [17]. Large projects worked by multiple developers are generally not feasible for the system to be divided into isolated modules for each developer [18]. This framework enables an offline collaboration with shelf modelling tools and online scenarios by offering a web based modelling front-end. Offline collaboration enables collaborators to check out artifacts from VCS and commit local changes to the VCS repositories in an asynchronous long transaction [17]. In order to extend the traditional VCS, the MONDO COLLABORATION FRAMEWORK advances secure collaborative modelling features as follows:

- Integration with VCS;
- Fine-grained model-level access control;

- Property-based locking;
- Automated model merge; and
- Offline and online collaboration.

3.3. Software changes

Brindescu et al. provided a thorough analysis in an article entitled *How Do Centralized and Distributed Version Control Systems Impact Software Changes* [6]. In the article, the authors stated that developers have little knowledge of whether they are benefitting from the use of DVCS. The purpose of this study is to look at the influence of DVCS in terms of splitting, grouping, and committing changes. 820 participants were recruited for a survey and 409m lines of changed codes were analyzed in 358300 commits. It was found that different approaches of VCS have different effects on developers, teams, and processes.

3.4. Software branching

Another article considered was written by Barr et al., *Cohesive and Isolated Development with Branches* [7], that discussed how the DVCS branching process allows developers to collaborate on tasks in highly cohesive branches while enjoying reduced interference from any developers working on other tasks even though the tasks are strongly coupled to theirs. According to the authors, Open Source Software (OSS) projects are rapidly adopting DVCS because their project histories are easier to understand when faced with maintenance tasks and they are easier to revert to a previous state if the branch created has a problem [7]. In order to understand how DVCS branches protect developers from interruptions, an evaluation was done on how branches are used and the benefit that could be gained by examining the Linux history. There are three principal contributions which are as follows:

- Convincing evidence from the study of sixty projects that branching and undistributed has led to the rapid adoption of DVCS;
- Two new measures were defined which are branch cohesion and distracted commits; and
- The cohesiveness of branches and the effective isolation they provide against the interruptions intrinsic to concurrent development that has been quantified by applying the two new measures.

3.5 Open Source Software Projects

Another article that was studied is from Rodriguez-Bustos and Aponte, *How Distributed Version Control Systems Impact Open Source Software Projects* [10]. DVCS has technical facilities or features that assist software developers to work in new ways [10]. In this paper, a preliminary study was conducted to evaluate the impact of the migration process on a developer's organization and its contribution by analyzing Mozilla repositories that were migrated from CVS to Mercurial in 2007. Below is the list of aspects that were addressed during the analysis process:

- The authorship of change sets;
- The size and contribution of core developers; and
- The type of contribution based on the number of modified files before and after migration.

3.6 Curriculum Development

Another article focusing on learning Git is *Employing Git in the Classroom* by Kelleher [5], which discussed the introduction of Git as a mechanism to disseminate class exercises and submission of continuous assessments. As indicated by the author, the industry demands graduates with the knowledge of VCS [5]. For this study, second and third year computer science students were the target audience for Git. All students were signed up for a free Github user account as Github offers a free educational account which provides unlimited private repositories for the lecturer. The author claimed that computer science faculty's students should be encouraged to adopt Git as a standard mechanism to manage both class materials and continuous assessment submissions.

In addition, based on the article by Mandala and Gary [8], courseware development is like software development because there has been an important shift in courseware requirements determined by continuous changes, disruption caused by online technologies, and changes in learners' content consumption behaviors. The authors stated that the Configuration Management patterns for DVCS are as follows:

- Branch per developer;
- Feature separation through named branches;
- Working off named stable bases; and
- Other configuration management workflow patterns.

The systems help teachers and other educators to share their course materials with each other. Just like the OSS development, they can also participate in collaborative course development that can lead to better course materials.

4. Advantages of VCS

This section gives an overview of the advantages that VCS offers. VCS has been proven to accelerate and simplify the software development process. There are loads of advantages in using VCS for software projects. VCS allows people to work absolutely freely with the team. They can work on any file at any time without overlapping each other's work by writing over other people's code. If two developers make changes in the same file, the VCS will merge the changes or warn them that some of the codes are conflicting [25] because VCS has the ability to track each alteration or changes made in the files or codes [24]. The latest or the whole project will be in the VCS. Obviously, since VCS enables collaboration so people can share data, files, and documents more easily through the use of VCS.

Saving a version of the project after making changes is an important task. This task becomes tedious and quite confusing if people are not using VCS. Every time people commit changes, they create a new version of the corresponding file. VCS saves the version in a professional way. It does not need an extra folder or a complete copy of the project. When saving a new version, it also saves when those changes were made and by whom and also a message that describes the changes, so that the team will understand them later. When a certain version of the project is needed, software developers can request it at any time and they will have a snapshot of the complete project.

Another important aspect of the VCS is that it allows the older version of the code to be safely stored in the VCS repository. It means people can always go back to a specific version of the file. A VCS can be extremely useful because it will allow people to recover from accidental deletions or edits. For example, to go back to the previous version that was working perfectly and also to compare the previous version with the latest version and to see what has been changed. VCS repositories contain important information that can be used during the development [20].

The VCS also enables people to conserve and save disk space on both the source control client and on the server. This is because it centralizes the management of the version. Thus, instead of having many copies spread around, there is only a central point where these copies are stored. VCS often uses algorithm to store all the changes. Therefore, many versions of the project can be kept without taking too much space.

5. 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.

References

1. Otte S. Version control systems. *Computer Systems and Telematics* 2009.
2. De Alwis B, Sillito J. Why Are Software Projects Moving From Centralized to Decentralized Version Control System? *In Proceedings of The 2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering* 2009; 36-39.
3. Collin-Sussmen B, Fitzpatrick BW, Pilato CM. Version Control For Subversion For: Subversion 1.7. Version Control With Subversion. 2002.
4. Cavalcanti G, Accioly P, Borba P. Assessing semistructured merger in version control systems: A replicated experiment. In *Proceedings of the 9th International Symposium on Empirical Software Engineering and Measurement 2002, ESEM'15.*
5. Kelleher J. Employing Git in the Classroom. *In Computer Application and Information Systems (WCCAIS), 2014 World Congress on*; 1-4.
6. Brindescu C, Codoban M, Shmarkatiuk S, Dig D. How Do Centralized and Distributed Version Control Systems Impact Software Changes? *In Proceedings of the 36th International Conferernce on Software Engineering* 2014.
7. Barr ET, Bird C, Rigby P, Hindle A, German D, Devanbu P. Cohesive and isolated development with branches. *Fundamental Approaches to Software Engineering* 2012; 316-331.
8. Mandala S, Gary KA. Distributed version control for curricular content management. *In Frontiers in Education Conference, IEEE* 2013*;* 802-804.
9. Tichy W F. RCS – A System for Version Control. *Software: Practice and Experience* 1985; 637-654.
10. Rodrigues-Bustos C, Aponte J. How Distributed Version Control Systems Impact Open Sources Software Projects. *In Proceedings Ninth IEEE Working Conference Mining Software Repositories* 2012*;* 36-39
11. Somasundaram R. *Git Version Control for Everyone*. UK: Birmingham; 2013.
12. Rigby PC, Barr ET, Bird C, German DM, Devambu P. What Effect Does Distributed Version Control Have on OSS Project Organization? *In Proceedings of the 1st International Workshop on Release Engineering* 2013; 29-32.
13. Cavalcanti, G. What Merge Tool Should I Use?. *In Proceedings Companion of The 2017 ACM SIGPLAN International Conference on Systems, Programming, Languages, and Applications: Software for Humanity* 2017; 19-20.
14. Apel S, Lessenich O, Lengauer C. Structured Merge with Auto-Tuning: Balancing Precision and Performance. *In Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (ASE'12)* 2012.
15. Case DM, Eloe NW, Leopold JL. Scaffolding Version Control into the Computer Science Curriculum. *In Proceedings of the 2016 International Workshop on Distance Education Technology (in conjunction with the 22nd International Conferennce on Distributed Multimedia Systems (DMS'16))* 2016.
16. Alam O, Sousa V, Syriani E. Towards Collaborative Modelling Using a Concern-Driven Version Control System. In *Proceedings of MODELS 2017 Satellite Event: Workshops (ModComp, ME, EXE, COMMitMDE, MRT, MULTI, GEMOC, MoDeVVa, MDETools, FlexMDE, MDEbug), Posters, Doctoral Symposium, Educator Symposium, ACM Student Research Competition, and Tools and Demonstrations co-located with ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS)* 2017.
17. Debreceni C, Bergman G, Bur M, Rath I, Varro D. The MONDO COLLABORATION FRAMEWORK: Secure Collaborative Modeling over Existing Version Control Systems. *In Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering* 2017; 984-988.
18. Kelly S. Collaborative Modelling with Version Control. *In Federation of International Conferences on Software Technologies: Applications and Foundations* 2017*;* 20-29.
19. Rao NR, Sekharaiah KC. Methodological Review Based Version Control System with Evolutionary Research for Software Processes. *In Proceedings of the 2nd International Conference on Information and Communication Technology for Competitive Strategies*. 2016; 14.
20. Greene GJ, Fischer B. Interactive Tag Cloud Visualization of Software Version Control Repositories. *In Software Visualization (VISSOFT), 2015 IEEE 3rd Working Conference* 2015; 56-55.
21. Chacon S, Straub B. *Pro Git*. USA: Berkeley; 2014.
22. Yip A, Chen B, Morris R. Pastwatch: A Distributed Version Control System. In Proceedings of the 3rd Conference on Networked Systems Design & Implementation NSDI'06 2006; **3**: 29.
23. Aslan K, Skaf-Molli H, Molli P. Coonecting Distributed Version Control Systems Communities to Linked Open Data. 2012 International Conference on Collaboration Technologies and Systems (CTS) 2012; 242 – 250.
24. Rocha FG, Menezas PM, Nascimento CD, Junior CR, Oliveira AAD. Continuous Integration and Version Control: A Systematic Review. 14th Conference on Information Systems & Technology Management – CONTECSI 2017; **14.**
25. Spinellis D. *Tools of the trade.* 2005; 108-109.