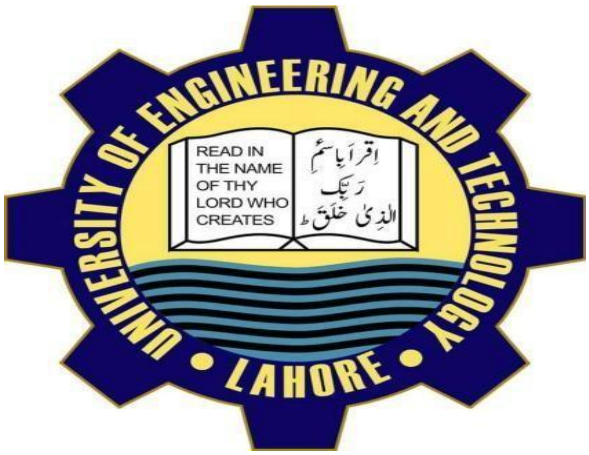
Version Control Strategies

(PROFESSIONAL PRACTICES IN SOFTWARE DEVELOPMENT)



SUBMITTED TO:

SIR UMAR QASIM

SUBMITTED BY:

Aaiza Shehzadi (2021-SE-05)
Ayesha Sharif (2021-SE-40)
Nimra Iqbal (2021-SE-31)
Mohsina (2021-SE-35)
Numan Danish (2021-SE-49)

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ENGINEERING AND TECHNOLOGY
LAHORE (KSK CAMPUS)

Table of Contents

1. Abstract:

Version Control Systems (VCS) have been used by many software developers during project developments as they help them to manage the source code 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.

2. 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 (RCS), Software Configuration Management, Source Code Management (SCM), and Source Code Control (S, 2009). 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. The 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. It is an approach of managing the tasks of multiple team members who are sometimes at different locations.

3. 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). 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 used CVCS tools are CVS and Subversion. Since the emergence of DVCS tools such as Git, Mercurial, Bazaar, and Bit Keeper, many open and closed source projects have been proposing to move or have already moved their source code repositories to a DVCS. Summary of the 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, Bit Keeper
Software Characteristics that suitable	Projects that allow only several users to contribute to the software development.	DVCS is suitable for a single or more developers because the project repository is distributed to all the developers and this ability offers a great improvement for the projects.

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 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.

4. 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. 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 enables the developers to work cooperatively. It stores a master copy of files history and to read, retrieved, commit new changes to a certain version, they need to contact a server.

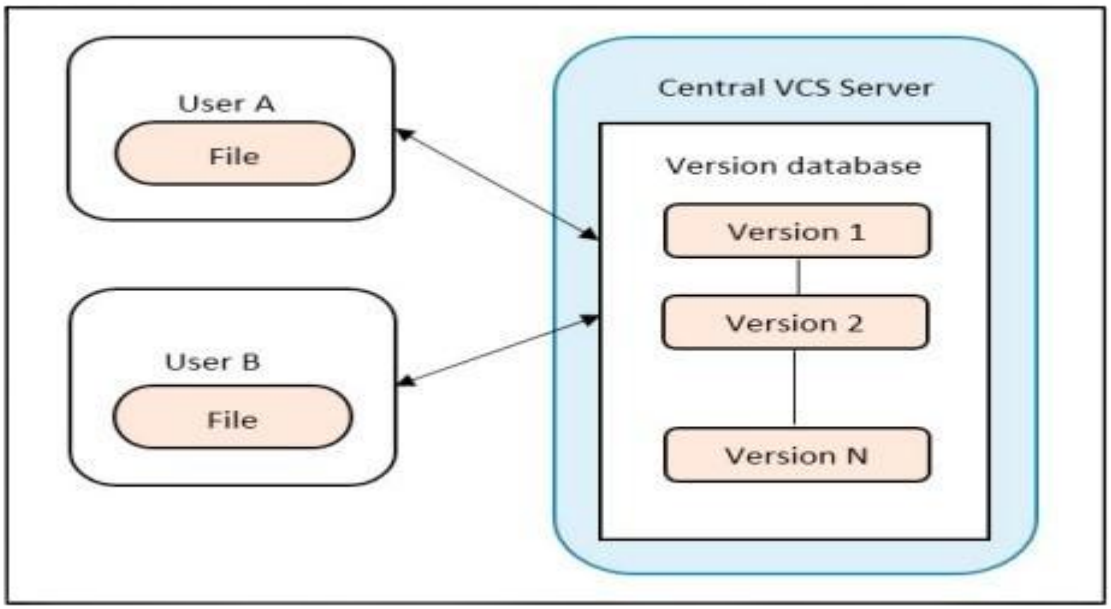


Fig1. Centralized version control (CVC)

As shown in Fig. 1, CVCS has a single repository which is basically the server. 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 is retrieved. As the files are stored in the server, it means that any changes made will automatically be 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. Despite that, this restriction will affect the participation and authorship of new contributors because 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.

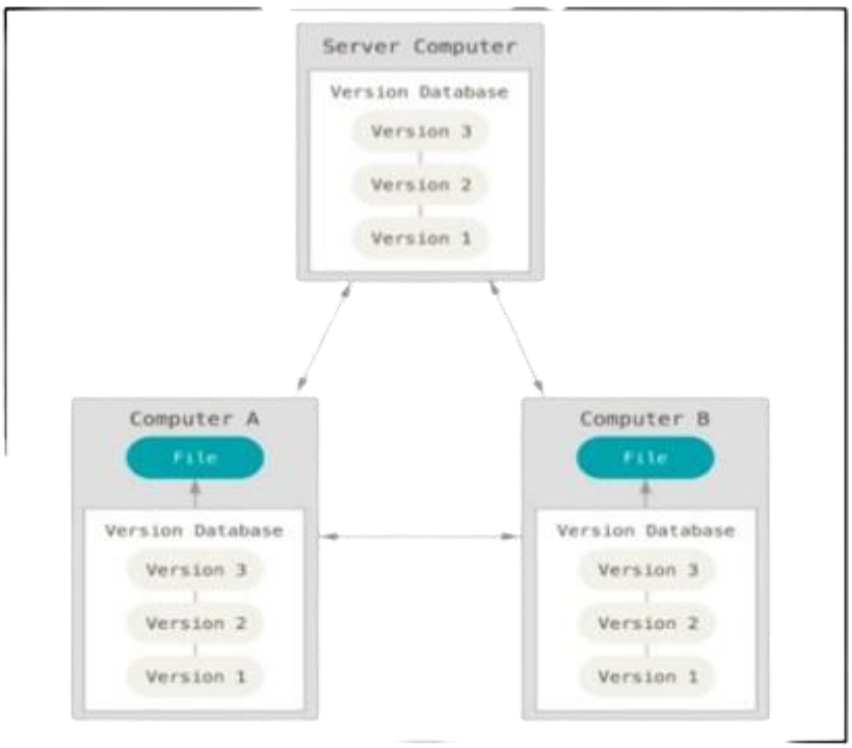
From a passive user, the developers will become advanced users if they become familiar with the project. At this stage, the developers are allowed to send some messages to the mailing list. This activity can be tracked; Developers who have partial knowledge about the project may do some report bugs, 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 written access to the code repository of the project. Developers who have this writing permission can commit any changes made; and
The last process 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 used CVCS are Concurrent Version System (CVS), Subversion (SVN), and perforce.

5. Distributed Version Control Systems (DVCS)

Nowadays, Open-Source Software (OSS) projects largely adopt the DVCS. 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 developers are using CVCS, it will work well if they can contact the server, but if they cannot contact the server, it will be a big problem for them. 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. (R., 2013)



1. Fig 2. Distributed Version Control Systems (DVCS)

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 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. 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 investigate 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 the server to recover it. Every clone is genuinely a full backup of all the data. Examples of DVCS tools are Git, Bazaar, Mercurial, Bit keeper, and Darcs. Bit keeper and Bazaar are two of the earliest DVCS tools. Git and Mercurial are the newest DVCS tools that improve the merging and branching features.

2. Application of CVS

Numerous research has 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.

2.1 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. As the authors pointed out, the conflict resolution strategies adopted by version control systems can be classified into three areas which are unstructured, structured, and semi-structured. Semi-structured approach is the combination of the strength of both, the expressiveness of the structured merger, and the generality of the unstructured merge 4. The purpose of the semi-structured approach is to be responsible for providing 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 to identify false positive or spurious conflicts 13. According to the writer, unstructured merger 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.

2.2 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 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 "7. 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. 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. 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
- Offline and online collaboration

2.3 Software changes

Bradesco 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.

2.4 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. 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.

2.5 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
- The type of contribution based on the number of modified files before and after migration.

2.6 Curriculum Development

Another article focusing on learning Git is Employing Git in the Classroom by Kelleher, 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. 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 (Mandala S, 2013; 803-804), 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.
- 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.

7 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 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 because VCS can track each alteration or changes made in the files or codes 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 using 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 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 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.

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.

8 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. 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

S, O. (2009). *Version control systems Computer Systems and Telematics.* lahore: otte.

Mandala S, G. K. (2013; 803-804). *Distributed version control for curricular content managemen.* lahore: Mandala S, Gary

R., S. (2013). *Git Version Control for Everyone.* UK : Birmingham.