Exploring the Dynamics of Version Control Systems: An In-depth Analysis

ARUN KUMAR N V1, BASAVARAJ2, DANESH KADAPPA HOSUR3, DILIP RATHOD4, DR. SRINIVASA A H[5]

[1, 2, 3, 4, 5] *Dr. Ambedkar Institute of Technology, Bengaluru, Karnataka, India*

Abstract*— Managing the source code of the project and other related documents in an organization is a mandatory need, which may ensure clarity in the delivery of the product enhancing the focus of the organization towards its intended product's quality. We have a plethora of software configuration management tools at our disposal in this digital age of computing to manage different documents, their revisions, versions, and so on. This article examines the significance of different Version Control Systems (VCS) that have developed to support software development lifecycles, compares popular VCS products available on the market based on features, and evaluates their effectiveness across selected attributes. Also, we suggest a new tool bearing few of the best face raise in our comparison study as well as a few extra attributes that we believe will raise the quality of this new tool. This finishes can combat the issues we challenge existing finishes concerning business.

I. INTRODUCTION

In the ever-evolving landscape of software development, managing codebase changes and facilitating seamless collaboration among developers are paramount challenges. Version Control Systems (VCS) emerge as a fundamental solution to these challenges, providing a structured and efficient approach to tracking modifications, coordinating team efforts, and safeguarding the integrity of software projects.

A Version Control System serves as a repository for source code and project files, enabling developers to work collaboratively without the risk of conflicting changes. By meticulously recording alterations, it offers a historical perspective on the evolution of a project, facilitating not only error correction but also strategic decision-making regarding feature development and software releases.

Version Control Systems, emphasizing their pivotal role in modern software engineering. From the basic principles of capturing changes and creating a version history to the more advanced features such as branching and merging, VCS provides a robust framework for maintaining order in the collaborative development process. As we navigate through the complexities of distributed teams, parallel development, and the imperative for agile methodologies, understanding and implementing effective version control practices become imperative. As complex teams are involved in the modern software development, it is very frequent for many versions of the same software to be deployed in different locations or systems and for them to be working synchronously on its development. Most of the times only a specific part of the program does contain bugs or features (by the fixing of some problems and the introduction of others as the program develops). Therefore, to locate and to fix bugs, it is of absolute important to be able to retrieve and run various versions of the software to determine in which version(s) the problem occurs. It may also be necessary to develop different parts of the software in parallel (for instance, where one version which has all its bugs fixed, but no new features (branch), while the different adaptation is place new features, that is still under experiment are processed on this project.

At a basic level, we can keep multiple copies of different versions of the program and label them accordingly, which is a form of manual version control. This method was previously used in large operating system projects. Although feasible, it requires a significant amount of work and is highly inefficient. It demands a great deal of discipline and can introduce new errors by the developer. Since the codebase remains largely unchanged, it also necessitates the creation of an admin user to control

who can configure which version, ensuring the codebase is not compromised. This adds a considerable amount of complexity. To address these issues, we have various VCS that separate version control management from the users' perspective.

Furthermore, in various fields beyond software development, it has become commonplace for a single document to be managed by a team whose members may not be physically co-located and may have different responsibilities. A Version Control System (VCS) that can track and document ownership of revisions to documents would be highly beneficial in such scenarios.

There are a lot of version control systems available these days, but the earlier ones required more mental effort to grasp and had a higher learning curve because they were many modern version control systems (VCS) reduce the complexity by offering a user-friendly command-line interface (CLI) with comprehensive documentation and built-in help (e.g., Git). Additionally, some VCS tools provide straightforward graphical user interfaces (GUIs) or integrate seamlessly with the system's default file explorer (e.g., Subversion and TortoiseSVN).

These tools are available in two architectural types: centralized and distributed. Centralized systems depend on a single repository for storing most of the project data and versioning information, with client interfaces retrieving only the latest version for use. In contrast, distributed systems—currently more popular—store a complete copy of the repository on each user's machine, facilitating independent and offline work.

II. RELATED WORK

We intentional several adaptation control systems common now which cover various approaches that went in construction them in the way that centralized vs distributed architectural types. We have also included some older tools in the study to identify how far their modern counterparts have come regarding performance, usability and interoperability. Specifically, we studied the following eight tools:

- Source Code Control System
- Revision Control System (RCS)
- Concurrent Version System (CVS)
- DVCS
- Subversion
- GNU Bazaar
- Git
- Mercurial

A. Revision Control System (RCS)

RCS, individual of the very first VCS, that happened into life in 1982 by Walter F. Tichy who was from Purdue University [1]. RCS is now kept up for one GNU Project. RCS was initially developed for software development, but it is also useful for text documents which need frequent revision. RCS can only do single file operation[2]. As of now, it does not have a provision to support atomic commit. It supports branching though for individual files, but the syntax of its operation is hard to handle. Besides utilizing arms, teams favor to use the included locking mechanism and bother a sole head arm. It has got straightforward to understand the structure. However, the main con being its operational limitation is restricted to only one user at a time. No concurrency. It is only capable of working locally. This tool can only support the waterfall model of the software development workflow[3-5]. No GUI alternative is available.

B. Concurrent Version System (CVS)

CVS uses a client-server architecture model[6]. Released under the GNU GPL. It was very popular with open-source projects in its early days. No new releases since 2008 or put in other words, is dead. CVS uses a client-server architecture. Client and server may be run on the same machine for local development. Supports concurrent development where each developer edits his/her working copy and checks-in the changes to the server. All files are versioned accompanying any that is incremented accompanying further check-ins. We have got a feature which allows the users to different versions, a complete history of changes can be viewed and analyzed, besides being able to check out an old snapshot of the project sorted by a given date or as of a revision number. CVS servers can allow "anonymous read access". Can maintain different branches for a project [7]. CVS can store various versions of the same file using a unique feature known as delta compression. CVS does not consider the move and renaming as separate versions.

GUI clients such as Tortoise CVS and Smart CVS available.

C. Source Code Control System

In the early days of software development, managing code changes was a cumbersome and error-prone process. Developers relied on manual tracking and notation systems, often leading to confusion and lost versions. This is where Marc J. Rochkind's "Source Code Control System" (SCCS), introduced in 1972, emerged as a revolutionary solution. SCCS, developed at Bell Labs, was a version control system designed specifically for tracking modifications in source code and other text files. It offered a powerful and innovative way for programmers to collaborate and maintain a clear history of their work. At its core, SCCS functioned by storing each revision of a file as a "delta," a set of changes applied to the previous version. This approach minimized storage space by only saving the differences between versions. When needed, SCCS could reconstruct any historical version by applying deltas in sequence. One of SCCS's defining features was the automatic inclusion of an "sccsid string" within the source code itself. This allowed developers to easily navigate back in time, retrieve older code, and analyze the evolution of their software. The first publicly available version, SCCS v4, released in 1977, marked a significant milestone. This version introduced a text-based history file format, replacing the earlier binary formats and enhancing user readability. This shift made it easier for developers to understand and manage the version control history.

D. Subversion

Subversion, developed by CollabNet, is a centralized revision control system. It is open-source under the Apache license and is the second-most popular VCS after GIT. Currently, it is an Apache-funded top-level project. Prominent organizations such as SourceForge, Apache Software Foundation, GCC, Mono, and FreeBSD use SVN for source control. Subversion supports branching and tagging, facilitating non-linear workflows. Unlike other VCS, branching is not a costly operation. Any change, even minor, retains the full revision history of files. Commits are true atomic operations. It has a native client-server, layered library design, and offers language bindings for various programming languages. Subversion provides a single source of access control and supports tracking features, including change lists to organize commits into groups. Mature user interfaces like TortoiseSVN are available, and online hosting options are offered by Deveo, Assembla, and RiouxSVN [14-16].

E. GNU Bazaar

GNU Bazaar, developed by Canonical, the team behind Ubuntu, is a distributed and client-server system. It suits individual developers managing multiple branches of local content [17-19]. Crafted in Python, it's cross-platform, free, and open-source. Embraced by major players like Linux Foundation, Ubuntu, MySQL, Debian, MariaDB, and more, it offers genuine branching at low cost. Whether in central or distributed mode, it excels in code management, with seamless migration from Subversion due to CLI similarity. Its CLI is intuitive, complemented by extensive documentation and built-in GUI tools. Highly adaptable to various workflows, it performs remarkably well even on sluggish networks with substantial revision histories. Renaming tracking for files and directories is provided, sans the necessity of a dedicated server. Boasting over 100 plugins and a Python API for custom development, it integrates smoothly with Launchpad for online hosting [20].

F. Git

Git, an advanced version control system (VCS), enables working on files and facilitating collaboration among multiple individuals while tracking changes. Linus Torvalds initially developed Git in 2005 to support the development of the Linux kernel. It operates under the GNU General Public License version 2.0 and is presently the most widely adopted VCS, both within companies and among standalone developers. As of 2016, it commands a significant 70% of search interest among VCSs and generates the most questions per day on Stack Overflow. Backed by a robust open-source community, Git offers comprehensive support for MS Windows and Unix-like systems. It accommodates both linear and nonlinear development and is compatible with traditional (waterfall) and modern (agile) software models. Git simplifies complex operations like branching, committing, undoing, checkpointing, and merging, making it invaluable for intricate software projects. With its distributed repository type, Git

ensures the cryptographic integrity of every aspect of a software project through checksums on files and commits. For users unfamiliar with Git's workflow, it supports various workflows such as Dictator and Lieutenants, Integration Manager, and Subversion-style workflows. It suits projects of all sizes, from kilobytes to exabytes, across any IDE, offering an easy-to-use command-line interface. Popular online hosting providers for Git repositories include GitHub, BitBucket, and GitLab..

G. Mercurial

Mercurial, developed by Matt Mackall and written in Python, is among the most popular version control systems (VCS) in use today [24-28]. It boasts a decentralized, fully-distributed architecture and is favored by prominent companies such as Facebook, Mozilla, Nginx, and NetBeans, as well as projects like Mozilla and Octave. With its cross-platform compatibility, Mercurial facilitates collaborative development without the need for centralized permissions management. Each commit is identified by a unique hexadecimal string generated through cryptographic hashing. Its command-line interface (CLI) is intuitive, and there are numerous graphical user interface (GUI) clients available, including the user-friendly TortoiseHg. Additionally, Mercurial offers a variety of online hosting options, such as CodePlex, Assembla, BitBucket, and RhodeCode.

H. DVCS

Traditionally, developers relied on Centralized Version Control Systems (CVCS) where a single server stores the entire codebase. This centralized repository serves as the sole source of truth, with developers checking out and checking in files to keep track of changes. Dependence on a central server could lead to bottlenecks, particularly with slow network connections. Additionally, a single point of failure – server downtime – could disrupt development. DVCS flips the script entirely. Instead of a central repository, each developer has a complete copy of the codebase, including its entire history, on their local machine. This local repository acts as a miniature version of the central server in CVCS. The magic lies in the peer-to-peer nature of DVCS. Developers can exchange changes and synchronize their local repositories with a remote repository without relying on a single central server.

III. PROPOSED SYSTEM

This paper proposes a novel Version Control System (VCS) that incorporates the key strengths identified through a comparative study of existing VCS solutions. The proposed system also introduces features that address the limitations of current workflows and reduce reliance on third-party tools. A rigorous analysis was conducted on existing VCS tools, focusing on the following key parameters

1. Security
2. Storage Model
3. Ease of Use
4. Ease of development

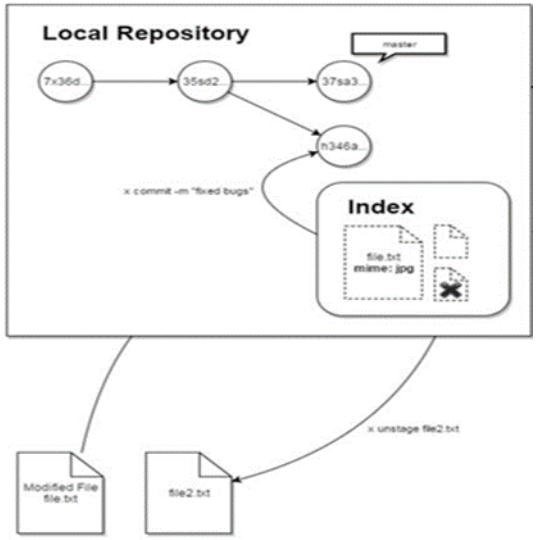

Fig. 2.1. Architecture diagram for the proposed tool.

A. Security

Security This section addresses the security of the revision history, encryption of revisions, and recently discovered vulnerabilities.

Secure Communication and Authentication: Secure communication and user authentication are paramount for protecting the integrity of the revision history. We propose HTTPS as the primary protocol for remote operations due to its:

- Security: HTTPS encrypts data in transit, safeguarding user information and content from unauthorized access.
- Efficiency: HTTPS offers efficient data transfer, minimizing communication overhead.

- Compatibility: HTTPS is widely supported by existing applications, eliminating the need for custom protocol implementations.

While SSH (Secure Shell) is also a secure option, its complexity can be a barrier for some users. HTTPS and OAuth, a widely adopted authorization framework, provide a simpler and more user-friendly approach with comparable security features. Additionally, the adoption of HTTPS/2 further enhances both speed and security of communication.

SHA-2 Encrypted Commits: To ensure the integrity of committed data, our system calculates SHA-2 checksums for files during the commit process. SHA-2 is a robust and currently uncompromised cryptographic hash function, safeguarding commit messages during transfer.

Deprecation of SHA-1: Recognizing the theoretical vulnerabilities of SHA-1 (deprecated by NIST in 2011), we opt for the more secure SHA-2 algorithm. This approach mitigates the risk of compromised revisions due to outdated encryption methods.

B. Storage Model

This section addresses storage considerations, including change set/snapshot management, disk space utilization, and binary file handling.

Enhanced Binary File Support: Current VCS tools often treat all files equally, regardless of type. This approach can be inefficient for binary files, as operations like diffs are slower compared to text files. While existing algorithms address this, lacking knowledge of specific binary formats hinders performance.

Our proposed solution involves adding file type information as attributes during commit. This allows for implementing specialized algorithms for common formats (e.g., image, audio) for faster operations. This approach increases VCS size but offers significant performance gains. We plan to support common formats initially, with an extensibility mechanism for developers to add custom logic for specific needs.

Additionally, current tools often load entire binary files into memory for diffs, leading to performance issues with larger files. We propose splitting large files or deltas into smaller chunks for efficient diffing and combining the results. This strategy improves performance when handling large binary files and repositories.

SHA-2 Encrypted Commits: As discussed previously, SHA-2 encrypted commits offer security benefits while also ensuring data integrity. This means the system can verify data during transmission and reception, eliminating potential data corruption. As long as the internal files remain unaltered, the system maintains complete awareness of file presence and history, even without direct knowledge of the content within those files.

C. Ease of Use

This section evaluates user experience (UX) factors, including command-line interface (CLI) clarity.

Clear and Consistent CLI: While Git dominates the version control system (VCS) market, its CLI presents challenges. Discussions reveal user dissatisfaction with its complexity, particularly for those transitioning from simpler tools like SVN. Inconsistencies in command naming and functionality further contribute to confusion. For instance, resetting files requires distinct commands for single files and entire directories . This ambiguity is a hurdle for both beginners and experienced users.

Our proposed tool addresses these issues by offering a streamlined, modern interface with clear and memorable commands. We prioritize explicitness over implicitness, even if it necessitates additional commands. Distinct commands like x unstage file.txt and x reset file.txt enhance clarity.

Cross-platform Support: Subversion excels in user-friendliness due to its mature user interface options. While CLI proficiency offers long-term benefits, a user-friendly interface caters to novice users. Our solution prioritizes a user-friendly experience across platforms. The architecture allows developers to create plugins, potentially offering an online repository for user-driven features (similar to the binary file support extension concept).

Platform-specific Filesystem Integration: Tools like TortoiseSVN provide valuable UX through integration with file explorers. This allows users to perform VCS operations directly within the file explorer. Color-coded icons further enhance the user experience by visually indicating the repository state.

D. Ease of Development

This section would discuss the factors that make the proposed tool easier to develop for contributors. Here are some potential aspects to consider including:

- Modular Design: The codebase could be designed with modularity in mind, allowing for independent development and maintenance of different functionalities. This would make it easier for developers to contribute specific features or bug fixes without needing to understand the entire system.
- Open Source: Consider making the tool open-source. This would allow the developer community to contribute code, identify and fix bugs, and propose improvements.
- Detailed Documentation: Comprehensive documentation that explains the codebase, design choices, and contribution guidelines would be helpful for new developers to get started.
- Testing Framework: A robust testing framework would ensure the quality and stability of the codebase. This would make it easier for developers to write unit tests and integration tests for their contributions.
- Continuous Integration: Implementing a continuous integration (CI) system would automate testing and deployment processes, providing developers with faster feedback on their changes.
- Version Control System Integration: Utilizing a version control system like Git would allow developers to track changes, collaborate effectively, and revert to previous versions if necessary.

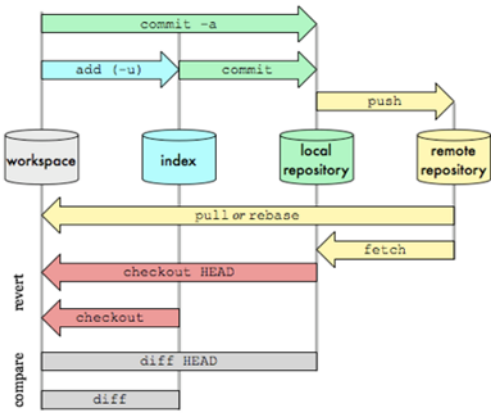

Fig 3.1 Data flow diagram.

CONCLUSION

This paper explored the concept of version control systems (VCS) by examining their key characteristics: repository structure, concurrency control mechanisms, storage management, and data unit definition. We also reviewed the historical development of VCS and their fundamental functionalities.

There are several directions for research in version control systems. One field of research maybe making version control vacant to expansive type of rules by exploring the likelihood of mixing version control feature directly into development languages, or perhaps fortifying the growth surroundings in the form of a foundation, such that developers could easily create applications with built-in version control capabilities. This could be considered as bringing version control to the context of the application. While our analysis identified Git as the current market leader, followed by Apache Subversion, Mercurial, and Perforce Helix (for enterprise use), each tool possesses its own limitations. Recognizing these limitations, this paper proposes a novel VCS tool that integrates the most advantageous features from existing solutions, while introducing additional functionalities to support modern development workflows. The proposed tool represents a promising avenue for future implementation and further feature development.

REFERENCES

[1] RCS GNU Project - https://www.gnu.org/software/rcs/

[2] Tichy, W. F. (1985). RCS—a system for version control. Software: Practice and Experience, 15(7), 637-654.

[3] RCS tutorial - http://archive.oreilly.com/pub/a/perl/excerpts/system- admin-with-perl/five-minute-rcs-tutorial.html

[4] RCS a system for version control- http://dl.acm.org/citation.cfm?id=4202

[5] RCS in brief http://jodypaul.com/SWE/RCSTutorial/RCSTutorial.html

[6] CV wikipedia - https://en.wikipedia.org/wiki/Concurrent_Versions_System

[7] Using CVS - https://www.ibm.com/support/knowledgecenter/SSSHYH_7.1.0.6/com.ibm.netcoolimpact.doc/admin/imag_selecting_CVS_version_manager.html

[8] What is perforce helix - https://www.perforce.com/versioning-engine

[9] [Perforce manual - - https://www.perforce.com/perforce/r15.2/manuals/dvcs/

[10] Introducing helix - https://www.perforce.com/blog/150303/introducing-helix

[11] Perforce helix - https://en.wikipedia.org/wiki/Perforce_Helix

[12] Enterprise perforce - https://softwareengineering.stackexchange.com/questions/85845/why-big-companies-use-perforce

[13] Apache subversion wikipedia - - https://en.wikipedia.org/wiki/Apache_Subversion

[14] Apache subversion - https://subversion.apache.org/

[15] Subversion wikipedia - https://en.wikipedia.org/wiki/Subversion

[16] Version control with subversion - http://svnbook.red-bean.com/

[17] Bazaar - http://bazaar.canonical.com/en/

[18] Bazaar vs git - http://wiki.bazaar.canonical.com/BrzVsGit

[19] Bazaar stackoverflow - http://stackoverflow.com/questions/14926774/what-is-the-state-of- bazaar-version-control

[20] GNU bazaar - https://en.wikipedia.org/wiki/GNU_Bazaar

[21] Git documentation - https://git-scm.com/doc

[22] Github git - https://github.com/git/git-scm.com

[23] Git wikipedia - https://en.wikipedia.org/wiki/Git

[24] Mercurial - https://www.mercurial-scm.org/

[25] Mercurial wikipedia - https://en.wikipedia.org/wiki/Mercurial,http://stackoverflow.com/questions/35837/what-is-the-difference- between-mercurial-and-git

[26] [Managing with mercurial - https://www.ibm.com/developerworks/aix/library/au-mercurial/

[27] Mercurial vs git technical aspects - https://www.atlassian.com/blog/software-teams/mercurial-vs-git-why- mercurial

[28] Bitkeeper - http://www.bitkeeper.com/

[29] Junqueira, Daniel C, Bittar, Thiago J and Fortes, Renata P s.1 , A fine-grained and flexible version control for software artifacts : ACM, 2008. pp. 185--192.Bitkeeper github - https://github.com/bitkeeper-scm/bitkeeper

[30] Supporting distributed collaboration through multidimensional software configuration management. Chu-Carroll, Mark C and Wright, James. s.1. : Springer-Verlag, 2003. pp. 40--53

[31] Linux and bitkeeper - https://www.linux.com/news/bitkeeper-and-linux-end-road

[32] Best version control for a standalone computer - http://stackoverflow.com/questions/138621/best-version-control-for-lone-developer

[33] G2crowd best version control - https://www.g2crowd.com/categories/version-control-systems

[34] Choosing the best version control - https://www.codeproject.com/Articles/431125/Choosing-a-Version-Control-System-A-Beginners-Tour

[35] Version control options - https://www.sitepoint.com/version-control- software-2014-what-options/

[36] Rhodecode survey - https://rhodecode.com/insights/version-control-systems-2016

[37] Popular version control systems - https://rhodecode.com/insights.com