S.R.M. Nagar, Kattankulathur, Chengalpattu

District MAY 2024

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

BUILDING VERSION CONTROL SYSTEM

MINI PROJECT REPORT

18CSE316J - ESSENTIALS IN CLOUD AND DEVOPS

Submitted by

VISWESH M[RA2111056010016]
MONIKA[RA2111033010158]
RISABH[RA2111033010105]
ASHRAYA[RA2111033010149]

Under the guidance of
Dr. R. Logeshwari
Assistant Professor, Networking and Communications

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING

of

FACULTY OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR – 603 203

BONAFIDE CERTIFICATE

Certified that Mini project report titled **"BUILDING VERSION CONTROL SYSTEM"** is the bonafide work of **VISWESH M [RA2111056010016], MONIKA MARADANA [RA2111033010158], RISABH RAI [RA2111033010105], ASHRAYA AGNIHOTRI [RA2111033010149]** who carried out the minor project under my supervision. Certified further, that to the best of my knowledge, the work reported here does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

Dr. R Logeshwari

SUPERVISOR
Assistant Professor
Department of Networking
and Communication,
SRMIST

ABSTRACT

Version Control Systems (VCS) are fundamental tools utilized by software developers to manage source code, enabling the preservation of project versions and facilitating collaboration. As a cornerstone of software engineering, VCS streamlines the management, organization, and coordination of development endeavors. Collaboration among software developers is essential for project success, and VCS plays a pivotal role by providing a collaborative framework that enhances teamwork and productivity. This paper provides an in-depth exploration of the background and related research surrounding VCS, shedding light on its significance and impact in the realm of software development. By synthesizing existing knowledge and ideas, this paper aims to contribute to a deeper understanding of VCS and its implications for effective project management and collaboration. Through a comprehensive review of literature and case studies, key concepts, methodologies, and best practices in VCS implementation are elucidated, providing insights into its role in ensuring project stability, scalability, and maintainability. Additionally, emerging trends and future directions in VCS research and development are discussed, offering valuable perspectives for further exploration and innovation in this critical domain of software engineering..

SRM Institute of Science and Technology

Own Work Declaration Form

Degree/Course : B.Tech CSE with specialization in Data Science

Student Details :
MONIKA MARADANA [RA2111033010158]
M VISWESH [RA2111056010016]
RISABH RAI [RA21110330105]
ASHRAYA AGNIHOTRI [RA2111033010149]

We hereby certify that this assessment compiles with the University's Rules and Regulations relating to Academic misconduct and plagiarism, as listed in the University Website, Regulations, and the Education Committee guidelines. We confirm that all the work contained in this assessment is our own except where indicated, and that we have met the following conditions:

- Clearly references / listed all sources as appropriate
- Referenced and put in inverted commas all quoted text (from books, web, etc.)
- Not made any use of the report(s) or essay(s) of any other student(s) either past or present
- Acknowledged in appropriate places any help that I have received from others (e.g. fellow students, technicians, statisticians, external sources) • Compiled with any other plagiarism criteria specified in the Course handbook / University website
- I understand that any false claim for this work will be penalized in accordance with the University policies and regulations.

TABLE OF CONTENTS

CHAPTER – 1

INTRODUCTION

1. Motivation:

Motivation is the driving force behind the adoption and continual refinement of Version Control Systems (VCS) in software development practices. As the complexity and scale of software projects continue to evolve, the need for robust systems to manage source code becomes increasingly pronounced. Without a centralized system for version control, developers face the daunting prospect of tracking changes manually, leading to inefficiencies, errors, and potential conflicts. Moreover, in an era characterized by distributed teams and remote collaboration, the need for a cohesive platform that enables real-time synchronization and coordination becomes paramount. VCS not only fulfills this need but also empowers developers to work autonomously on parallel branches, fostering innovation and experimentation within the development workflow.

This project delves into the motivation driving the adoption and study of VCS in software development. By examining the historical background, theoretical frameworks, and practical applications of version control, we aim to elucidate the significance of VCS in contemporary software engineering practices. Through a comprehensive analysis of existing literature and case studies, we endeavor to uncover the underlying principles and best practices that underpin effective version control strategies. Additionally, we explore emerging trends and future directions in VCS research, offering insights into potential avenues for innovation and advancement in this critical domain.

2. Objective

Motivation is the driving force behind the adoption and continual refinement of Version Control Systems (VCS) in software development practices. As the complexity and scale of software projects continue to evolve, the need for

robust systems to manage source code becomes increasingly pronounced. Without a centralized system for version control, developers face the daunting prospect of tracking changes manually, leading to inefficiencies, errors, and potential conflicts. Moreover, in an era characterized by distributed teams and remote collaboration, the need for a cohesive platform that enables real-time synchronization and coordination becomes paramount. VCS not only fulfills this need but also empowers developers to work autonomously on parallel branches, fostering innovation and experimentation within the development workflow.

This paper delves into the motivation driving the adoption and study of VCS in software development. By examining the historical background, theoretical frameworks, and practical applications of version control, we aim to elucidate the significance of VCS in contemporary software engineering practices. Through a comprehensive analysis of existing literature and case studies, we endeavor to uncover the underlying principles and best practices that underpin effective version control strategies. Additionally, we explore emerging trends and future directions in VCS research, offering insights into potential avenues for innovation and advancement in this critical domain.

1.3. Problem Statement

Developing a robust version control system (VCS) to manage the evolution of software projects is critical for collaborative development. Current solutions like Git and Subversion have limitations such as complex branching strategies, scalability issues, and lack of certain features necessary for specific project requirements. Additionally, the learning curve for these systems can be steep for new users.

Proposed Solution

The project aims to create a modern version control system that addresses the shortcomings of existing solutions while introducing innovative features to streamline development workflows. The proposed system will prioritize

simplicity, flexibility, and scalability, making it suitable for projects of all sizes and complexities.

Key features of the proposed solution include:

1. **Intuitive User Interface:** A user-friendly interface that simplifies common version control tasks, reducing the learning curve for new users.
2. **Flexible Branching Model:** Support for both centralized and distributed branching models, allowing teams to choose the workflow that best fits their project requirements.
3. **Performance and Scalability:** Optimized performance to handle large repositories and accommodate growing project sizes without sacrificing speed.
4. **Advanced Conflict Resolution:** Enhanced conflict resolution mechanisms to minimize merge conflicts and facilitate smoother collaboration among team members.
5. **Extensibility:** A plugin architecture that enables developers to extend the functionality of the VCS to meet specific project needs or integrate with other tools and services.
6. **Enhanced Collaboration:** Built-in tools for code review, issue tracking, and project management to facilitate seamless collaboration among team members.

1.4. Challenges

Building a version control system (VCS) involves overcoming various challenges to ensure it functions effectively and reliably. Here are some of the main challenges:

Efficient Storage: Version control systems need to efficiently store different versions of files. This requires balancing the need for compact storage with the ability to quickly access and retrieve versions when needed. Techniques like delta encoding (storing only the differences between versions) and compression are commonly used to optimize storage.

Concurrency: VCS must handle concurrent access from multiple users or processes. This involves managing concurrent writes to the repository while maintaining data consistency and integrity. Techniques like file locking, optimistic concurrency control, and transaction management are used to address this challenge.

Conflict Resolution: When multiple users make conflicting changes to the same file or codebase, the VCS must be able to detect and resolve these conflicts automatically or with user intervention. Conflict resolution algorithms need to balance preserving user changes with minimizing data loss and maintaining code integrity.

Performance: VCS should provide fast operations for common tasks such as committing changes, checking out branches, and merging code. Optimizing performance involves efficient data structures, indexing, and algorithms to handle large repositories and complex operations quickly.

Scalability: As projects grow in size and complexity, the VCS needs to scale to handle large repositories, numerous users, and a high volume of concurrent operations. Scalability challenges include optimizing server infrastructure, network bandwidth, and data transfer protocols to support growing demands.

Branching and Merging: Effective support for branching and merging is essential for managing parallel development efforts, feature branches, and release cycles. VCS must provide robust tools and algorithms for creating, managing, and merging branches while preserving code history and minimizing conflicts.

User Interface and User Experience: A well-designed user interface is crucial for the usability and adoption of a VCS. The system should provide intuitive commands, informative feedback, and visualizations to help users understand the status of their repository, changes made, and conflicts encountered.

Security: VCS should ensure the security and confidentiality of code and sensitive information stored in repositories. This includes access control mechanisms, authentication, encryption of data in transit and at rest, and protection against common security threats such as unauthorized access, data breaches, and malware.

Extensibility and Integration: VCS should support integration with other development tools and workflows, such as issue trackers, continuous integration (CI) systems, and code review platforms. Extensibility APIs and plugins enable developers to customize and extend the functionality of the VCS to suit their specific requirements.

Compatibility and Interoperability: VCS should be compatible with a wide range of operating systems, development environments, and programming languages. Interoperability with other version control systems and standards ensures seamless collaboration and migration between different tools and platforms.

Addressing these challenges requires a combination of software engineering principles, algorithms, data structures, and system architecture design tailored to the specific requirements and constraints of version control systems. Additionally, continuous testing, monitoring, and feedback from users are essential to identify and address issues as they arise and evolve over time.

CHAPTER 2

LITERATURE SURVEY

Version control systems (VCS) are crucial tools in modern software development, enabling efficient collaboration, tracking changes, and ensuring the integrity and stability of software projects. Over the years, numerous version control systems have been developed, each offering unique features and addressing specific needs of software development teams. In this literature survey, we explore the evolution, features, and key advancements in version control systems.

1. **Centralized Version Control Systems (CVCS)**: CVCS, exemplified by Concurrent Versions System (CVS) and Subversion (SVN), were among the earliest version control systems. In CVCS, a single central repository stores the entire history and files of a project. While CVCS simplifies collaboration by providing a single source of truth, it suffers from drawbacks such as a single point of failure and the need for constant network connectivity.
2. **Distributed Version Control Systems (DVCS)**: DVCS, represented by Git and Mercurial, revolutionized version control by decentralizing the repository. Each developer has a complete local copy of the repository, allowing for offline work and faster operations. Git, in particular, gained widespread adoption due to its speed, flexibility, and powerful branching and merging capabilities. Mercurial, while less popular, offers similar features and is preferred by some development teams.
3. **Feature-Based Version Control Systems**: Feature-based version control systems, such as Plastic SCM and Perforce, focus on managing features or changesets rather than individual files or revisions. These systems enable more granular control over project changes and facilitate parallel development of features by different team members.
4. **Cloud-Based Version Control Systems**: With the rise of cloud computing, several version control systems have migrated to the cloud, offering benefits such as scalability, accessibility, and built-in collaboration tools. Examples include GitHub, GitLab, and Bitbucket, which provide hosting services for

Git repositories along with issue tracking, code review, and continuous integration (CI) capabilities.

5. **Specialized Version Control Systems**: Certain version control systems are tailored for specific use cases or domains. For instance, Fossil integrates version control, bug tracking, and wiki capabilities into a single package, making it suitable for small projects or individual developers. Similarly, version control systems like Plastic SCM emphasize graphical user interfaces and support for large binary files, catering to the needs of game development and multimedia projects.

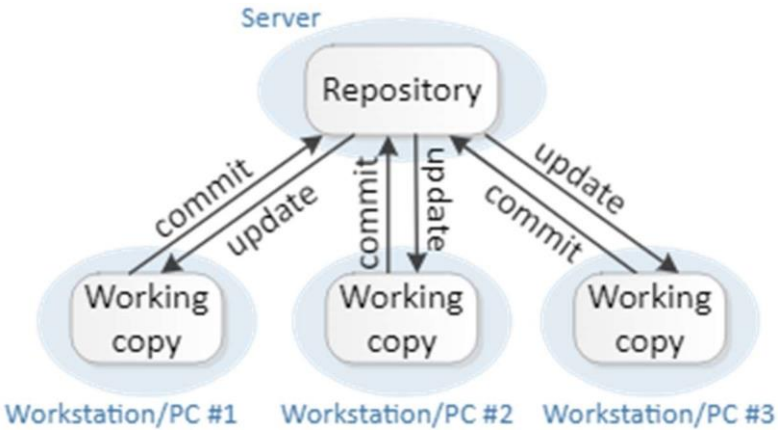
6. **Advancements and Trends**: Recent advancements in version control systems focus on enhancing scalability, performance, and collaboration features. Techniques such as shallow cloning, partial cloning, and repository virtualization aim to reduce the overhead of working with large repositories. Furthermore, integration with containerization technologies like Docker and Kubernetes enables versioning of infrastructure code and deployment configurations.

7. **Challenges and Future Directions**: Despite significant advancements, version control systems face challenges related to scalability, security, and usability. As software projects continue to grow in complexity and size, there is a need for improved techniques for managing dependencies, automating workflows, and ensuring data integrity. Future research directions may include leveraging machine learning algorithms for code analysis and automated conflict resolution, as well as exploring decentralized consensus mechanisms for distributed version control systems.

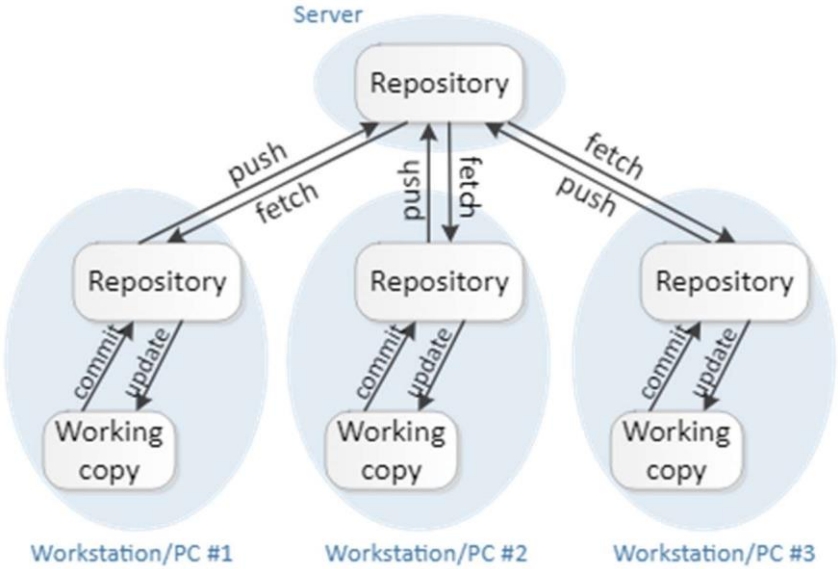
CHAPTER 3

SYSTEM ARCHITECTURE AND DESIGN

Centralized Version Control System:

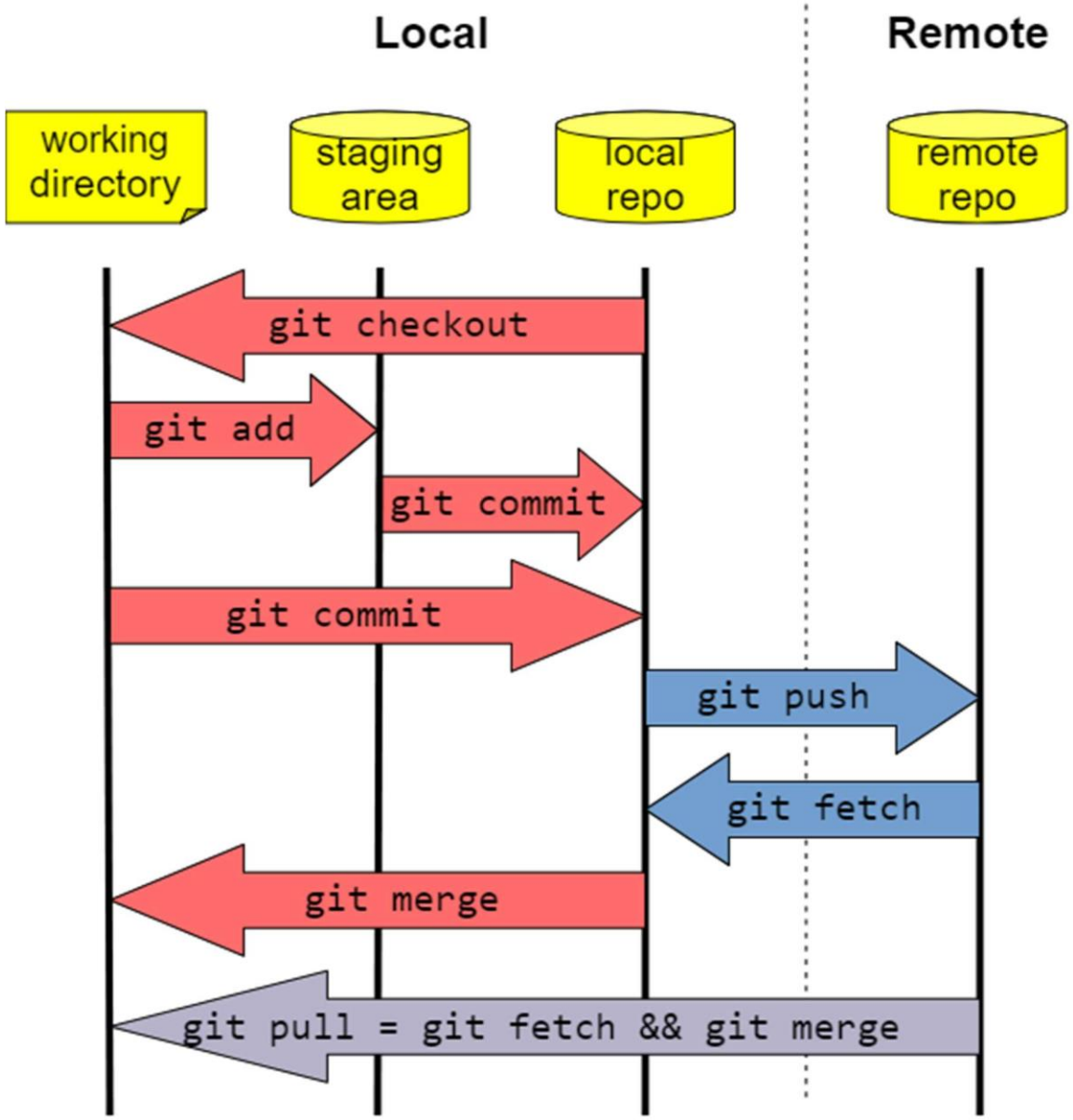


Distributed Version Control System:



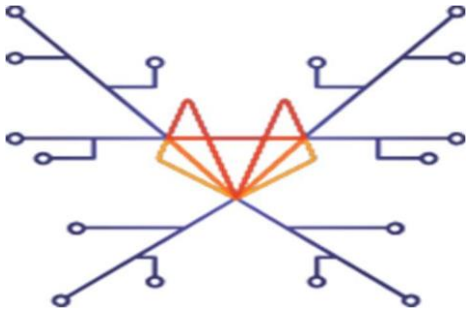
Operation:

The diagram at right shows which operations affect the working copy (red), which affect the repository (blue), and which affect both (purple). Merging, git add, and the staging area are explained later.



CHAPTER 4

IMPLEMENTATION



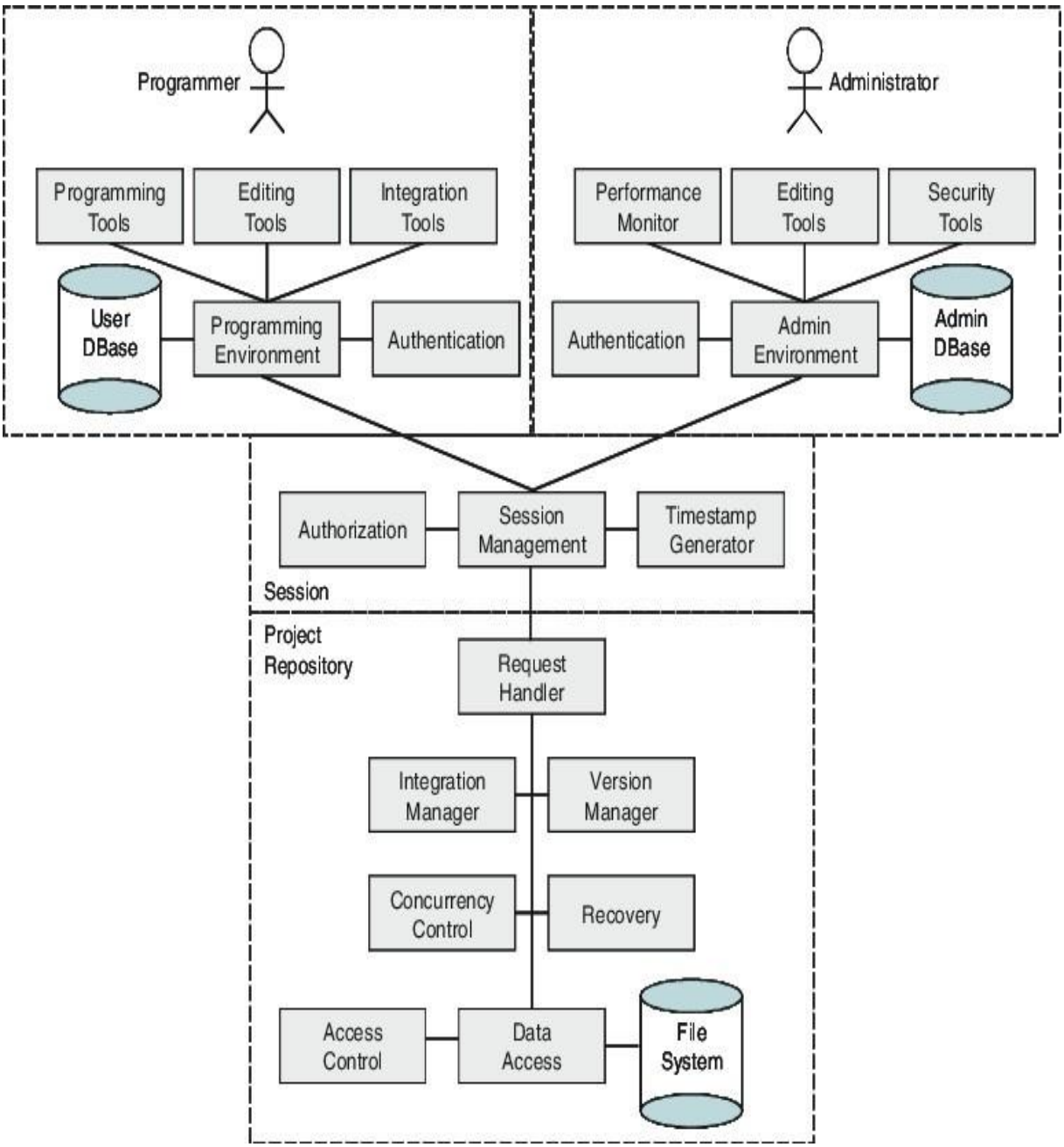
Building a version control system is a complex task that involves several components and steps. Here's a simplified breakdown of the implementation process:

1. **Requirement Analysis:**
    - Understand the basic functionalities required in a version control system such as repository management, version tracking, branching, merging, etc.
    - Determine the technologies and tools needed for development.
2. **Design Phase:**
    - Define the architecture of the system including the database schema, client-server communication protocol, and user interface.
    - Decide on the programming language(s) and frameworks to be used.
3. **Setting Up the Environment:**
    - Set up the development environment with necessary tools like version control software (e.g., Git), IDEs, and databases.
    - Create a new project repository for the version control system.
4. **Implementation:**
    - Develop the core functionalities such as creating a new repository, adding files, committing changes, viewing history, creating branches, merging branches, resolving conflicts, etc.

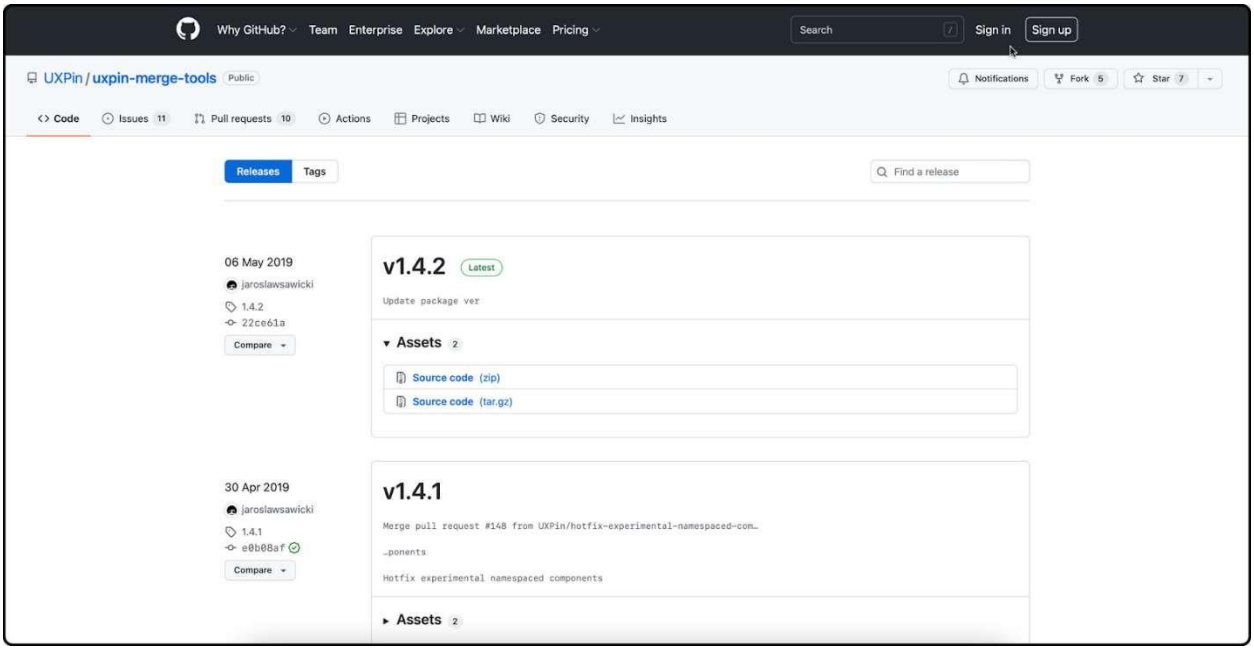
- Implement user authentication and access control mechanisms.
- Write tests to ensure the reliability and correctness of the system.

5. **Integration and Testing:**
    - Integrate different components of the system and ensure they work seamlessly together.
    - Conduct thorough testing including unit tests, integration tests, and system tests.
    - Address any bugs or issues that arise during testing.
6. **Documentation:**
    - Document the system architecture, APIs, and user guides for using the version control system.
    - Provide examples and tutorials to help users understand how to use the system effectively.
7. **Deployment:**
    - Deploy the version control system to a production environment.
    - Set up monitoring and logging to track system performance and usage.
    - Continuously monitor and update the system to ensure it remains stable and secure.
8. **Maintenance and Support:**
    - Provide ongoing maintenance and support to address user inquiries, fix bugs, and add new features.
    - Regularly update the system to incorporate improvements and security patches.

Concurrent Versioning System:



Version Control UI Design



CHAPTER 5

APPLICATIONS AND ADVANTAGES

1. 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.

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 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. 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. According to the writer, unstructured merge tools are entirely text-based so the tools will revolve the merge conflicts via textual similarity. In contrast, the structured merge tools which are based more towards programming languages, will use the programming syntax to resolve the merge conflicts. 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.

1.2. Software branching

Another article considered was written by Barr et al., *Cohesive and Isolated Development with Branches* , 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

CHAPTER 6

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.

In conclusion, version control systems play a fundamental role in modern software development, offering essential features for collaboration, change management, and project governance. The evolution from centralized to distributed systems, coupled with advancements in cloud computing and specialized tooling, reflects the dynamic nature of version control technologies and their continuous adaptation to the evolving needs of software development teams.

CHAPTER 7

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