IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 6, May 2026
Impact Factor: 8.2

Analytical Framework for Merge Conflict Assessment Using Git Repository Metrics

Sarthak Narayan Kadam and Rohini S. Kapse
M. Sc. (C.S)-II, Department of Computer Science and Applications, MVPS K. T. H. M. College, Nashik.
Assistant Professor, Department of Computer Science and Applications, MVPS K. T. H. M. College, Nashik
Corresponding Author: Name: Sarthak Narayan Kadam
kadamsarthak96k@gmail.com
https://orcid.org/0009-0000-3466-8621

Abstract: *As software development shifts toward highly distributed models, the frequency of merge conflicts has become a significant bottleneck in Continuous Integration (CI) pipelines. While existing research has largely focused on complex Machine Learning (ML) prediction models or high-level workflow comparisons, there remains a gap in lightweight, interpretable repository analytics. This paper proposes a framework for merge conflict assessment based on core Git metrics: code churn, branch lifetime, contributor overlap, commit density, and historical conflict rates. We introduce a risk assessment model designed for practical integration into DevOps workflows. Our preliminary analysis suggests that repository-level metrics can effectively signal integration risks without the computational overhead of speculative merging or deep learning architectures.*

Keywords: Git, Merge Conflicts, Repository Analytics, Code Churn, DevOps, Software Engineering

I. INTRODUCTION

Modern software development is characterized by high-velocity collaboration, facilitated by Distributed Version Control Systems (DVCS) like Git. These systems allow teams to work concurrently on disparate features; however, this parallelization necessitates a final integration phase that is frequently fraught with technical challenges.

The primary obstacle in this process is the merge conflict. When independent changes overlap textually or semantically, the integration complexity increases, leading to unstable repositories and significant coordination overhead. Research indicates that merge conflicts are not merely technical hurdles but socio-technical disruptions that waste developer time, delay releases, and introduce regressions.

The difficulty of integration scales non-linearly with branch lifetime. The longer a branch remains isolated, the further it diverges from the main branch. Furthermore, large decentralized teams create coordination challenges where simultaneous modifications to hotspot files become inevitable.

Despite the maturity of version control systems, a gap exists in the literature. Previous studies have focused predominantly on workflow comparisons or computationally intensive machine learning techniques for conflict prediction. There is a noticeable lack of practical, lightweight analytical frameworks that provide repository-level visibility into integration health.

This paper addresses this gap by proposing a Repository Analytics Framework. The proposed methodology quantifies integration risk through churn analysis, contributor overlap analysis, and branch lifetime evaluation. Our contribution is an interpretable conflict-risk assessment model that utilizes standard Git metadata to empower maintainers with actionable insights.

The remainder of this paper is organized as follows. Section 2 reviews the evolution of version control systems and existing conflict management research. Section 3 details the proposed framework and repository metrics. Section 4 describes the analytical methodology. Section 5 discusses observations and implications. Finally, Sections 6 and 7 conclude the paper and discuss future work.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 6, May 2026
Impact Factor: 8.2

II. LITERATURE REVIEW

2.1 Evolution of Version Control Systems

The transition from centralized systems such as SVN to distributed systems such as Git has significantly redefined collaborative software development. Git's architecture enables developers to create local branches and perform independent modifications before integration into shared repositories.

2.2 Git Workflows and Repository Management

Workflows such as Git Flow provide structured release management but often create long-lived branches that increase divergence. In contrast, GitHub Flow and Trunk-Based Development emphasize frequent integration to reduce repository instability.

2.3 Merge Conflicts and Repository Instability

Merge conflicts arise when concurrent modifications overlap textually or semantically. Excessive code churn, prolonged branch divergence, and simultaneous file modifications significantly increase integration complexity. Frequent conflicts often indicate unstable repository components and poor modularization.

2.4 Repository Mining and Software Analytics

Mining Software Repositories (MSR) has emerged as an important field for analyzing software evolution and collaboration behavior. Repository analytics can identify hotspot files, unstable modules, contributor interactions, and development bottlenecks.

2.5 Existing Conflict Prediction Techniques

Existing approaches for conflict prediction include heuristic techniques and machine learning-based models. Many ML approaches achieve high prediction accuracy but often behave as black-box systems that lack interpretability and require extensive computational resources.

III. PROPOSED SYSTEM

3.1 Framework Overview

The proposed framework is designed as a lightweight repository analytics pipeline consisting of five major stages:

1. Data Collection
2. Commit Analysis
3. Branch Analysis
4. Contributor Analysis
5. Risk Assessment

The framework extracts repository metadata using Git logs and APIs to estimate integration complexity and repository instability.

Data Collection → Commit Analysis → Branch Analysis → Contributor Analysis → Risk Assessment

Figure 1: Repository Analytics Framework

3.2 Repository Metrics

3.2.1 Code Churn

Code churn is defined as the number of lines added, deleted, or modified during development. Excessive churn often indicates unstable repository components.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 6, May 2026
Impact Factor: 8.2

3.2.2 Branch Lifetime

Branch lifetime represents the duration between branch creation and final integration. Longer lifetimes increase branch divergence and merge complexity.

3.2.3 Contributor Overlap

Contributor overlap measures the number of developers simultaneously modifying shared repository files. Higher overlap increases socio-technical coordination complexity.

3.2.4 Commit Density

Commit density measures the frequency of commits during active branch development. High commit density reflects rapid repository evolution.

3.2.5 Conflict Occurrence Rate

Conflict occurrence rate represents the historical frequency of merge conflicts within repository components.

3.3 Conflict Risk Assessment Model

The proposed framework introduces a lightweight analytical model for estimating merge conflict risk:

Where:

$$RF = \alpha \frac{C}{C_{max}} + \beta \frac{L}{L_{max}} + \gamma \frac{O}{O_{max}} \quad (1)$$

C represents normalized Code Churn

L represents normalized Branch Lifetime

O represents normalized Contributor Overlap

α, β, γ represent tunable weighting factors

The model emphasizes interpretability and practical deployment over computational complexity.

IV. ANALYTICAL METHODOLOGY

4.1 Repository Selection

The framework is conceptually evaluated using medium-scale open-source GitHub repositories selected according to the following criteria:

- Active development activity
- Multiple contributors
- Rich merge histories
- Frequent pull request activity

4.2 Tools Used

The proposed framework utilizes the following tools:

• GitPython
• GitHub API
• Python
• Pandas
• Matplotlib

4.3 Data Collection Process

Repository metadata such as commit history, timestamps, file modifications, contributor activity, and merge events are extracted using Git logs and GitHub APIs. These metrics are analyzed to estimate integration risk and repository instability.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 6, May 2026
Impact Factor: 8.2

V. RESULTS AND DISCUSSION

5.1 Analysis Findings

Preliminary observations indicate several important trends:

- Files with excessive churn were more frequently associated with merge conflicts.
- Long-lived branches demonstrated higher integration complexity due to repository divergence.
- Contributor overlap significantly increased conflict resolution difficulty because multiple developers were required to coordinate modifications.

The reported observations are preliminary and intended to illustrate the applicability of the proposed framework**.**

5.2 Repository Stability Assessment

Repositories utilizing frequent integration practices demonstrated lower estimated risk scores compared to repositories relying on prolonged branching strategies. These observations support the idea that continuous integration reduces repository instability and merge complexity.

5.3 Discussion

The proposed framework demonstrates that lightweight repository-level metrics can provide meaningful insights into integration bottlenecks without requiring computationally expensive machine learning infrastructures. The framework prioritizes interpretability and practical integration into DevOps workflows.

VI. LIMITATIONS

The proposed framework has several limitations. First, the current study relies on preliminary analytical observations rather than large-scale industrial validation. Second, the framework primarily utilizes repository metadata and does not perform semantic code analysis. Finally, the proposed analytical model relies on heuristic weighting and may require calibration for different repository environments.

VII. CONCLUSION

This paper presented a lightweight analytical framework for merge conflict assessment using Git repository metrics. The proposed framework utilizes code churn, branch lifetime, contributor overlap, commit density, and historical conflict occurrence to estimate repository integration risk. The study highlights how repository-level metrics can provide meaningful insights into integration complexity and repository instability without requiring computationally intensive predictive infrastructures. The framework emphasizes interpretability, low computational overhead, and practical applicability within DevOps-oriented development environments.

VIII. FUTURE SCOPE

Future work may involve:

- Developing real-time dashboards for repository monitoring
- Incorporating semantic code analysis using Abstract Syntax Trees (AST)
- Extending evaluation to large-scale industrial repositories
- Integrating repository risk assessment into CI/CD pipelines

REFERENCES

[1] Owhadi-Kareshk, M., Nadi, S., and Rubin, J., “Predicting Merge Conflicts in Collaborative Software Development,” arXiv preprint arXiv:1907.06274, 2019.
[2] Shen, B., and Meng, N., “Automatic Detection and Resolution of Software Merge Conflicts: Are We There Yet?” arXiv preprint arXiv:2102.11307, 2021.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 6, May 2026 Impact Factor: 8.2

[3] Pan, R. et al., "Can Program Synthesis be Used to Learn Merge Conflict Resolutions?" arXiv preprint arXiv:2103.02004, 2021.
[4] Nugroho, Y. S. et al., "How Different are Different Diff Algorithms in Git?" Empirical Software Engineering, 2020.
[5] Bird, C. et al., "The Promises and Perils of Mining Git Repositories," IEEE Working Conference on Mining Software Repositories, 2009.
[6] Zimmermann, T., and Nagappan, N., "Predicting Defects using Network Analysis on Dependency Graphs," ICSE, 2008.
[7] Driessen, V., "A Successful Git Branching Model," 2010.
[8] Forsgren, N., Humble, J., and Kim, G., Accelerate: The Science of Lean Software and DevOps, IT Revolution Press, 2018