

**Performance Of Software Engineer Due to GIT Using**

**Course name:** Software Engineering

**Course code:** CSE412

**Semester:** Spring 2024

**Section:** 04

**Group:** 6

|  |  |
| --- | --- |
| Name | ID |
| Tanni Roy | 2019-3-60-114 |
| Tropa Das | 2019-3-60-034 |
| Asadullah Hil Gaalib | 2019-3-60-086 |
| Rifah Tasnim Tandra | 2019-1-60-133 |

Submitted to

**Nishat Tasnim Niloy**

Lecturer  
Department of Computer Science & Engineering

# Submission Date: 7/6/24

.

Performance Of Software Engineer Due to GIT

Using

*Abstract*—The progress in software development over the past three decades has been remarkable, with new concepts, programming languages, and frameworks continually emerging. This paper explores the significance of software in everyday life and the necessity of advanced software testing methodologies to ensure reliable software products. It delves into the challenges of distributed software projects, such as collaboration difficulties, project awareness, and code conflicts, and examines how version control systems like GitHub can mitigate these issues. Additionally, the paper discusses the use of GitHub in software engineering (SE) classes to enhance collaboration and objectively evaluate individual contributions. A case study of 91 students in SE classes reveals that GitHub improves skills in configuration management and version control, though there are risks in using it for individual effort assessment. Furthermore, the paper investigates the impact of programming languages on software quality using a large GitHub dataset. The study finds that language design modestly affects software quality, with static typing and functional languages performing slightly better. However, process factors such as project size and team size have a more significant impact on software quality. The findings provide valuable insights for researchers and recommendations for educators on integrating GitHub in SE classes.

*Index Terms*—component, formatting, style, styling, insert

# I. INTRODUCTION

This explores several key topics in software engineering, focusing on the effects of programming language choice on software quality, the need for improved digital skills education, and the role of Learning Analytics (LA) in enhancing software engineering training.

Firstly, it discusses the ongoing debate about whether static or dynamic typing is more beneficial. Proponents of static typing argue it helps catch defects early in the development process, while advocates of dynamic typing believe it conserves developer resources by catching errors as they occur. Historically, these debates have relied on anecdotal evidence due to the difficulty of isolating the impact of language choice from other factors influencing software quality, such as code size, team size, and project maturity. To address this, the passage highlights a study using a large dataset from GitHub, which includes multiple projects in various languages, to empirically investigate how language features affect defect rates. The study employs a mixed-methods approach, combining text analysis, clustering, and visualization to support findings from quantitative regression analysis, thereby providing a practical understanding of how programming languages impact software quality.

Secondly, the passage touches on the global digital skills shortage and initiatives like the UK Institute of Coding (IoC) that aim to bridge the gap between the supply and demand of skilled software engineers. At the University of Manchester, a project uses LA to analyze student interactions with version control and continuous integration systems. This analysis aims to understand better how students learn these essential skills and to develop tools for feedback and assessment to support their learning. The ethical collection and use of student data are emphasized, ensuring compliance with data protection policies and minimizing the risk of re-identification. The project also recognizes that learning digital skills requires both understanding the tools and how interactions with these tools evolve throughout the learning process.

Previous research is reviewed to show that practical, scenario-based learning is crucial for mastering software engineering tools like Git and GitLab. The paper concludes by proposing a pipeline for LA, detailing steps from data extraction to analysis, and emphasizing the need to consider the human aspect of learning in addition to collecting performance metrics. This approach aims to provide significant insights into student learning behaviors and inform improvements in teaching and learning processes in software engineering education.

# II. OBJECTIVE AND RESEARCH QUESTIONS

The primary objective of this research is to assess the impact of incorporating GitHub as a collaborative tool in Software Engineering (SE) educational settings. By integrating GitHub into classroom activities, the study aims to determine whether this platform can significantly enhance students’ practical skills in configuration management, promote effective team collaboration, and facilitate accurate assessment of individual contributions in team projects. Furthermore, the research seeks to identify the potential challenges and limitations of using GitHub in educational environments, particularly in large classes with 60 or more students.

To comprehensively explore the impact of GitHub in SE education, the study is guided by the following research questions:

1. RQ1. Does enforcing GitHub enhance Software Engineer’ configuration management skills? o This question aims to evaluate whether the use of GitHub improves students’ abilities to manage and control software configurations effectively. It will assess students’ understanding and application of version control concepts and their proficiency in using GitHub’s features for managing code changes and project

versions

1. RQ2. What are the main benefits and limitations ofusing GitHub for code and assignment submission? o This question seeks to identify the advantages and challenges associated with using GitHub for submitting and managing coding assignments. The study will explore how GitHub facilitates or hinders the submission process, including aspects such as ease of use, accessibility, and the effectiveness of feedback mechanisms.
2. RQ3. How do Software Engineer utilize GitHub in SEclasses’ team projects? o This question focuses on understanding the patterns and behaviors of students when using GitHub for collaborative team projects. It will investigate how students interact with the platform, manage their workflows, communicate with team members, and resolve conflicts. The goal is to gain insights into the collaborative dynamics facilitated by GitHub.
3. RQ4. Can GitHub be used as a basis for evaluation ofindividual effort? o This question examines the feasibility and effectiveness of using GitHub’s tracking and analytics features to evaluate individual contributions within team projects. It will explore the metrics available on GitHub, such as commit frequency, pull requests, and code reviews, to determine if they provide a reliable basis for assessing individual effort and performance in a team setting.

# III. METHODOLOGY

This methodology should be tailored to the specific needs, processes, and tools used by your organization. It’s crucial to establish clear guidelines and communicate the evaluation criteria to the software engineers to ensure transparency and alignment with organizational goals.

## A. Data Collection

To gather data for evaluating the performance of software engineers due to Git usage, we will utilize Git log commands or Git analytics tools to extract commit data from the repository. This will include collecting metadata such as author name, email, commit timestamp, commit message, and file paths modified. Additionally, we’ll integrate with issue tracking systems to correlate commits with associated issues or tasks for better context.

## B. Commit Analysis

The next step involves analyzing the commits made by software engineers. We will calculate the frequency of commits per developer over different time periods to gauge activity levels and assess their commitment to the project. Furthermore, we’ll evaluate the quality of commit messages by checking adherence to standard formats and guidelines. This includes measuring the size and complexity of commits using metrics like lines of code added/removed, file changes, and code churn. Large or complex commits will be identified for additional review or modularization to enhance code management.

## C. Code Review

In this phase, we’ll focus on the code review process. By integrating Git with code review tools or platforms such as GitHub Pull Requests or Gerrit, we can facilitate systematic review processes. We will analyze the frequency and timeliness of code reviews performed by each developer to ensure timely feedback. Additionally, we’ll evaluate the quality of review comments, assessing their relevance, accuracy, and constructiveness in improving code quality. This includes identifying and addressing code smells, security vulnerabilities, performance issues, or other quality concerns during code reviews.

## D. Branch Management

Branch management plays a crucial role in effective collaboration and project management. We’ll analyze the branch structure and naming conventions used by developers to ensure consistency and clarity in version control. Furthermore, we’ll evaluate adherence to established branching models such as Git Flow for efficient collaboration and release management. Long-lived or stale branches will be identified, indicating potential technical debt or inadequate branch management practices. We’ll also assess the efficiency of merging branches and resolving conflicts to maintain code integrity and project stability.

## E. Collaboration and Communication

Effective collaboration and communication are essential for successful software development. We’ll analyze pull request discussions, comments, and interactions between developers to evaluate collaboration dynamics. This includes assessing the quality of communication and responsiveness in addressing feedback or concerns during code review processes. Additionally, we’ll evaluate the level of collaboration and knowledge sharing within the team to foster a culture of learning and improvement. Identifying and addressing any bottlenecks or communication gaps will be crucial to enhancing collaboration and productivity.

## F. Productivity and Efficiency

Productivity metrics will be calculated to assess the output and efficiency of software engineers. This includes measuring the number of commits, lines of code added/removed, and files modified per developer over specific time periods. We’ll analyze activity patterns to identify periods of high or low productivity and correlate them with project milestones or deadlines. Furthermore, we’ll assess the impact of commits on the codebase in terms of complexity, risk, and potential side effects. This will help ensure the ability to deliver highquality work within expected timelines and maintain project success.

## G. Continuous Intregation and Deployment

Integration with CI/CD tools or platforms such as Jenkins or GitLab CI/CD will be crucial for automating build and deployment processes. We’ll evaluate the success rate of builds and deployments triggered by commits to ensure the reliability of the integration pipeline. Additionally, we’ll analyze the integration quality to ensure consistent and reliable integration of changes across different environments. Assessing the handling of merge conflicts and the resolution strategies employed by developers will be essential to maintain code stability and project continuity.

## H. Reporting and Feedback

Finally, we’ll generate reports or dashboards to consolidate the collected metrics and insights for easy visualization and analysis. Individual feedback will be provided to software engineers, highlighting areas for improvement and recognizing exemplary practices. Team retrospectives or code review sessions will be conducted to discuss Git usage, collaboration, and process improvements. Identifying training needs or opportunities for improving Git skills, coding practices, or team dynamics based on the evaluation results will be crucial for continuous improvement.

# IV. TECHNOLOGIES USED IN GITHUB

Git as a Decentralized Version Control System (DVCS) Git is a decentralized version control system (DVCS) that supports distributed software development. It allows multiple developers to work on a project simultaneously without a central repository. Key technologies and concepts within Git include:

* Autonomous Nodes: Each developer’s repository is a full-fledged repository with complete history and full version tracking capabilities.
* Local Commit Capability: Developers can commit changes locally without requiring immediate synchronization with a central repository.
* Branching and Merging: Git enables multiple branches to be created from the main development line. Developers can work on different branches, which can later be merged, resolving any conflicts that arise during the process.
* Content-Addressed Storage: Git stores data as snapshots of the project at specific points in time. It uses SHA-1 hashes to name and identify objects within its database, ensuring integrity and uniqueness.
* Efficient Synchronization: Git optimizes the storage and synchronization of duplicate files across the project using content-addressable storage and efficient algorithms, although it may struggle with large binary files.

# V. GIT AS COLLABORATIVE PLATFORM

GitHub builds on Git by adding a web-based interface and additional features that enhance collaborative software development. Key technologies and features provided by GitHub include:

* Repository Hosting: GitHub hosts Git repositories, providing a centralized place where code can be stored, shared, and collaborated on by multiple developers.
* Branch Management Tools: GitHub provides intuitive tools for managing branches, including visualizations of branch structures and easy methods for creating, merging, and deleting branches.
* Pull Requests: GitHub’s pull request system allows developers to propose changes to the codebase. These changes can be reviewed, discussed, and merged by the core team, ensuring code quality and consistency.
* Issue Tracking: GitHub includes an issue tracker to manage bugs, enhancements, and other project-related tasks. Issues can be linked to commits and pull requests, creating a seamless workflow for tracking development progress and addressing bugs.
* Continuous Integration/Continuous Deployment (CI/CD): GitHub integrates with CI/CD tools like Jenkins, Travis CI, and GitHub Actions. These integrations enable automated testing and deployment processes, providing immediate feedback to developers when new code is merged.
* Social Coding Features: GitHub’s social features, such as notifications, discussions, and contributions tracking, enhance communication and collaboration among developers. Users can watch repositories, fork projects, and contribute through pull requests.
* Private Repositories: GitHub provides private repositories for enterprise users, ensuring that proprietary code remains secure while benefiting from GitHub’s collaborative features.
* Educational Tools: GitHub Education offers resources and tools for learning and teaching software development, including features for automated testing and immediate feedback for students.
* Community Engagement: GitHub fosters a community of developers who contribute to open-source projects. Users can follow projects, participate in discussions, and contribute code, making GitHub a vibrant ecosystem for software development.

# VI. DATASET

The dataset originates from the GHTorrent dump of October 11, 2014, and focuses on GitHub projects with significant historical records that transitioned to Continuous Integration (CI) using Travis-CI. The selection criteria included mainline projects written in popular programming languages (Ruby, Python, JavaScript, PHP, Java, Scala, C, and C++), each having at least 200 pull requests. Travis-CI was chosen over Jenkins due to its consistent data environment and complete build histories accessible through the Travis-CI API. The dataset includes metadata on closed pull requests, such as comment counts and titles, covering 918 projects (48.7This dataset is useful for analyzing the impact of CI adoption on software development, offering insights into development workflows, collaboration, and code quality.

GitHut - Programming Languages and GitHub

VII. RESULT

|  |  |
| --- | --- |
| **Section** | **Action** |
| Origin | GHTorrent dump dated 10/11/2014. |
| Languages | Ruby, Python, JavaScript, PHP, Java, Scala, C, and C++. |
| Selection Criteria | Main-line projects (not forks) <br> - At least 200 pull requests <br> - Projects where pull requests are integral to development |
| CI Services Analyzed | Travis-CI (due to its hosted nature, consistent environment, and complete build history available via API). Jenkins was not considered due to its self-hosted nature and inconsistent data storage. |
| Final Project Count | 918 projects (48.7% of initial 1,884 projects) using Travis-CI. |
| Data Collected | Metadata on closed pull requests from GHTorrent, including: <br> - Number of comments on pull requests <br> - Titles of pull requests. |
| Purpose | To analyze the impact of adopting Continuous Integration (CI) on software development workflows, collaboration, and code quality. |

# VIII. CONTRIBUTE TO SOCIETY

GitHub is a multifaceted platform that makes significant contributions to society across various domains:

1. Open Source Ecosystem: GitHub is a hub for opensource projects, facilitating global collaboration and democratizing access to code . 2. Educational Resource: It serves as a learning platform for students and educators, offering handson experience and resources for digital literacy and coding skills.

3. Innovation and Entrepreneurship: GitHub supports startups and innovation by providing tools for code management and collaboration, fostering a culture of entrepreneurship. 4. Community Building: It hosts a vibrant developer community through events, hackathons, and forums, enabling collaboration, knowledge sharing, and networking.

1. Professional Development: GitHub serves as a portfolio for developers, aiding in career advancement and skill development through contributions to various projects.
2. Social Impact Projects: Many projects on GitHub focus on addressing social issues, using technology for humanitarian efforts and environmental sustainability.
3. Corporate Social Responsibility: GitHub promotes diversity, inclusion, and sustainability in the tech industry, supporting initiatives and projects that contribute to social good.
4. Advancement of Research: Academics and researchers use GitHub for collaboration, sharing code, and publishing reproducible research, advancing knowledge across various fields.

In essence, GitHub’s contributions span education, innovation, community building, professional development, social impact, corporate responsibility, and research, making it a vital platform for societal progress.

# IX. CONCLUSION

GitHub is the leading web-based platform for hosting and collaborating on Git repositories. It has revolutionized software development by providing a user-friendly interface, powerful collaboration tools, and a vibrant community for open-source projects. GitHub has become indispensable for developers, fostering collaboration, transparency, and innovation in the software industry.