Empirical Software Engineering (2027) 32:18
https://doi.org/10.1007/s10664-026-10941-0

What do feature branches tell us about feature implementation in open-source projects?

Nitish Patkar[1] · **Aimen Fahmi**[2] · **Timo Kehrer**[3] · **Norbert Seyff**[1]

Received: 24 September 2025 / Accepted: 22 July 2026
© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2026

Abstract
Although "feature" is a common term across requirements engineering, product-line engineering, and agile methods, its operationalization in practice–especially through feature branches–has received little empirical attention. Feature branches are widely recommended on platforms like GitHub; yet, empirical evidence on how they are actually used is scarce. This study investigates feature-branch development practices in open-source software (OSS) projects. We conducted a large-scale empirical analysis of 27,103 feature branches from 3,388 popular GitHub repositories. Using branch-naming conventions, we identified feature branches and measured their lifespan, commit history, and commit-share distributions among contributors. In addition, we profiled contributors according to their opened issues, opened pull requests, number of followers, and total stars. Our analysis revealed that feature branches in open-source projects are long-lived, with a median lifespan exceeding two years, diverging from the short, agile iterations typically employed in software development. Commit intensity is low and irregular, averaging fewer than two commits per day. Contribution is highly unequal within feature branches: a small share of contributors accounts for most commits, revealing strong dependence on a small subset of developers. These findings have several practical implications. Our findings suggest that monitoring contribution concentration within feature branches– for instance, by computing the share of commits attributable to the top contributor or a Gini coefficient over the branch's commit history–can help maintainers identify branches where development knowledge is held by very few individuals. Likewise, rolling commit intensity–computable directly from commit timestamps–may serve as an early signal of branch stagnation in OSS projects. Moreover, these findings establish an empirical baseline for future studies, including analyses of proprietary repositories.

Keywords Feature implementation · Data-driven methods · Contributor analysis · Agile development · Software product lines · Requirements engineering · Open-source software · Commit activity analysis

Communicated by: Xin Xia.

Extended author information available on the last page of the article

Published online: 04 August 2026

1 Introduction

The concept of a "feature" is central to software engineering, yet its definition and implementation strategies vary across different domains, such as requirements engineering (RE), software product lines (SPL), and agile software development (ASD) (Classen et al. 2008; Apel and Kästner 2009; Berger 2015; Molenaar et al. 2022). From an RE perspective, Classen et al. define a feature as a collection of related requirements, domain properties, and specifications (Classen et al. 2008). Apel et al. highlight features as distinct units of functionality that satisfy requirements and provide potential configuration options (Apel and Kästner 2009). Agile methodologies often treat features more informally, mapping them to user stories or functional increments intended for rapid iteration. Despite these theoretical distinctions, our empirical knowledge of how features are developed in practice remains limited. Existing literature has primarily focused on what features **are** rather than how they are **implemented** and **maintained** in real-world software projects. For instance, do feature branches serve as short-lived, isolated efforts aligned with best practices in continuous integration/continuous delivery (CI/CD), or do they persist much longer, hinting at diverging practices from typical agile guidelines?

In a previous study, we have emphasized the potential of feature branches as a rich data source for understanding feature development (Patkar et al. 2025). When developers name a branch based on a feature they implement, they effectively annotate their work with an intended functional goal, making it easier to track relevant commits, contributors, and discussion threads over time. Yet, to date, no other large-scale study has systematically analyzed feature branches to estimate how closely real-world development practices match the conceptual ideals of software engineering, specifically agile workflows.

To bridge this gap, we conduct a follow-up study of feature branches in open-source repositories, analyzing 27,103 feature branches across 3,388 GitHub projects. Next to studying general key characteristics of feature branches in open-source development, we focus on the *contributor dimension* of feature-branch development and examine how collaborators create, maintain, and contribute to feature branches. Our investigation sheds light on the practical realities of feature implementation and reveals several surprising findings with important implications in open-source software (OSS) projects.

To explore these issues in depth, we formulate the following three guiding research questions:

> ***RQ1*** **How are feature branches used in open-source projects, and what are their key characteristics?** To understand how feature branches are used generally in open-source projects, and more specifically whether they align with agile and CI/CD best practices, we investigate several key aspects: (1) *branch prevalence*–quantifying how many feature branches repositories maintain concurrently (Section 3.1.1); (2) *branch lifespan*–examining how long branches remain active compared to typical sprint durations (Section 3.1.2); (3) *contributor count and its relationship to branch lifespan*–analyzing how many developers work on each branch and how their participation relates to branch duration (Section 3.1.3); (4) *commit volume and intensity*–measuring the total number of commits per branch and daily commit rates to assess development pace

(Sections 3.1.4 and 3.1.6); (5) *relationship between contributor count and commit volume*–examining how team size relates to the total commits produced on a branch (Section 3.1.5); and (6) *commit distribution across contributors*–analyzing how commits are spread among the developers working on each branch (Section 3.1.7) These aspects directly relate to CI/CD principles, which advocate for short-lived branches with frequent integration to minimize merge conflicts and maintain code quality (Agrawal and Chari 2007). Our findings reveal that many feature branches remain active far longer than typical agile sprint durations, with low daily commit intensity, suggesting substantial deviation from prescribed best practices and potential challenges with prolonged code isolation.

RQ2 **How is development work distributed among contributors within feature branches, and what patterns of contribution emerge?** To understand contribution patterns and team dynamics, we examine: (1) *contribution concentration*–measuring the concentration of commit activity (Section 3.2.1); (2) *high-performer prevalence*–quantifying how many contributors drive the development (Sections 3.2.2 and 3.2.3); (3) *performance consistency*–analyzing whether high performers maintain their productivity across multiple branches (Section 3.2.4); and (4) *repository size effects*–investigating how team composition varies with project scale (Sections 3.2.5 and 3.2.6). Our findings reveal that, on average per branch, roughly 10% of contributors produce about 63% of that branch's commits, with high performers consistently dominating across branches, raising concerns about knowledge concentration and bus factor risks.

RQ3 **Can contributors' broader GitHub profile metrics help predict or explain their feature-branch performance?** To identify signals that distinguish high and low performers, we compare contributors across: (1) *activity metrics*–opened issues and pull requests that reflect engagement levels (Section 3.3.1)); (2) *social metrics*–followers and following counts indicating community visibility (Section 3.3.2); and (3) *repository metrics*–number of public repositories, total stars, and forks reflecting contribution breadth (Section 3.3.3). We find that high performers exhibit significantly larger GitHub footprints than low performers. The above findings have several practical implications. First, the prevalence of long-lived feature branches indicates that many open-source projects diverge from standard agile or CI/CD patterns (Agrawal and Chari 2007). Second, the strong association between contributor count and extended branch life highlights the importance of effective communication and coordination, especially in large or distributed teams. Finally, the presence of a small group of "power contributors" confirms the well-known Pareto principle in open-source ecosystems, yet also raises questions about how to mitigate risks related to concentrated expertise and ownership. The remainder of this paper is organized as follows: In Section 2, we describe the data collection and curation pipeline. In Section 3, we outline our data analysis approach and conclude the key findings. In Section 4, we critically examine the results, discussing their implications and potential for practical application. In Section 5, we discuss threats to validity and the steps taken to mitigate them. In Section 6, we review the broader literature on feature engineering, software analytics, and open-source collaboration. Finally, Section 7 summarizes our conclusions and outlines limitations of the current study along with directions for future research.

2 Data collection and curation

Open-source software (OSS) repositories provide a rich and transparent source of development data, making GitHub a suitable platform for studying contemporary software development practices. Given GitHub's explicit recommendation to use branches for feature development (Github 2024), we focus on feature branches as first-class artifacts for analyzing feature implementation workflows.

This section describes the data collection and curation pipeline used in our study. Given the known challenges of mining GitHub data at scale (Kalliamvakou et al. 2016) and the importance of transparent dataset curation (Borges et al. 2016), we provide multiple datasets at different levels of granularity in our replication package, each tailored to a specific analysis. Rather than maintaining a single monolithic dataset, we employ a progressive filtering and specialization strategy to (i) limit computational overhead, (ii) enable targeted analyses aligned with individual research questions, and (iii) support reproducibility by exposing intermediate datasets. This approach builds upon established filtering criteria (Munaiah et al. 2017) while extending them with progressive specialization for branch-level and contributor-level analyses.

Figure 1 provides an overview of this pipeline. Additionally, Table 1 provides a consolidated overview of all the datasets and their key characteristics.

2.1 Hybrid data collection approach

Our data collection follows a hybrid strategy that combines SEART (Dabic et al. 2021) metadata with fine-grained data retrieved via the GitHub API.

First, SEART is used to obtain repository-level metadata, enabling efficient large-scale filtering without incurring API rate limits. Second, for repositories that pass the filtering criteria, we use the GitHub REST API via the PyGithub library to retrieve detailed branch-level and commit-level information, including branch names, commit histories, contributor identities, and timestamps.

This hybrid approach allows us to scale to hundreds of thousands of repositories while selectively collecting detailed data only where required. All subsequent datasets are constructed by progressively enriching SEART metadata with GitHub API data.

2.2 Overview of the data collection pipeline

We begin with a base dataset of 1,580,895 GitHub repositories obtained from the SEART GitHub Search platform.[1] This dataset is progressively filtered and enriched to construct six interrelated datasets spanning repository-level, branch-level, and contributor-level analyses.

First, we apply activity, and popularity filters to derive a *Filtered Base dataset* of 53,802 repositories. Second, we identify repositories that use feature branches based on branch naming conventions, and we extract branch-level metadata to create a *Feature-Branches dataset*. Third, we enrich a subset of these repositories with full commit histories and contributor information, resulting in the *Enhanced Feature-Branches pre-curation dataset (43,347 rows)*. After this enrichment stage, we apply global repository–branch deduplication to eliminate duplicate branch records, yielding the *Enhanced Feature-Branches dataset*

[1] https://seart-ghs.si.usi.ch/

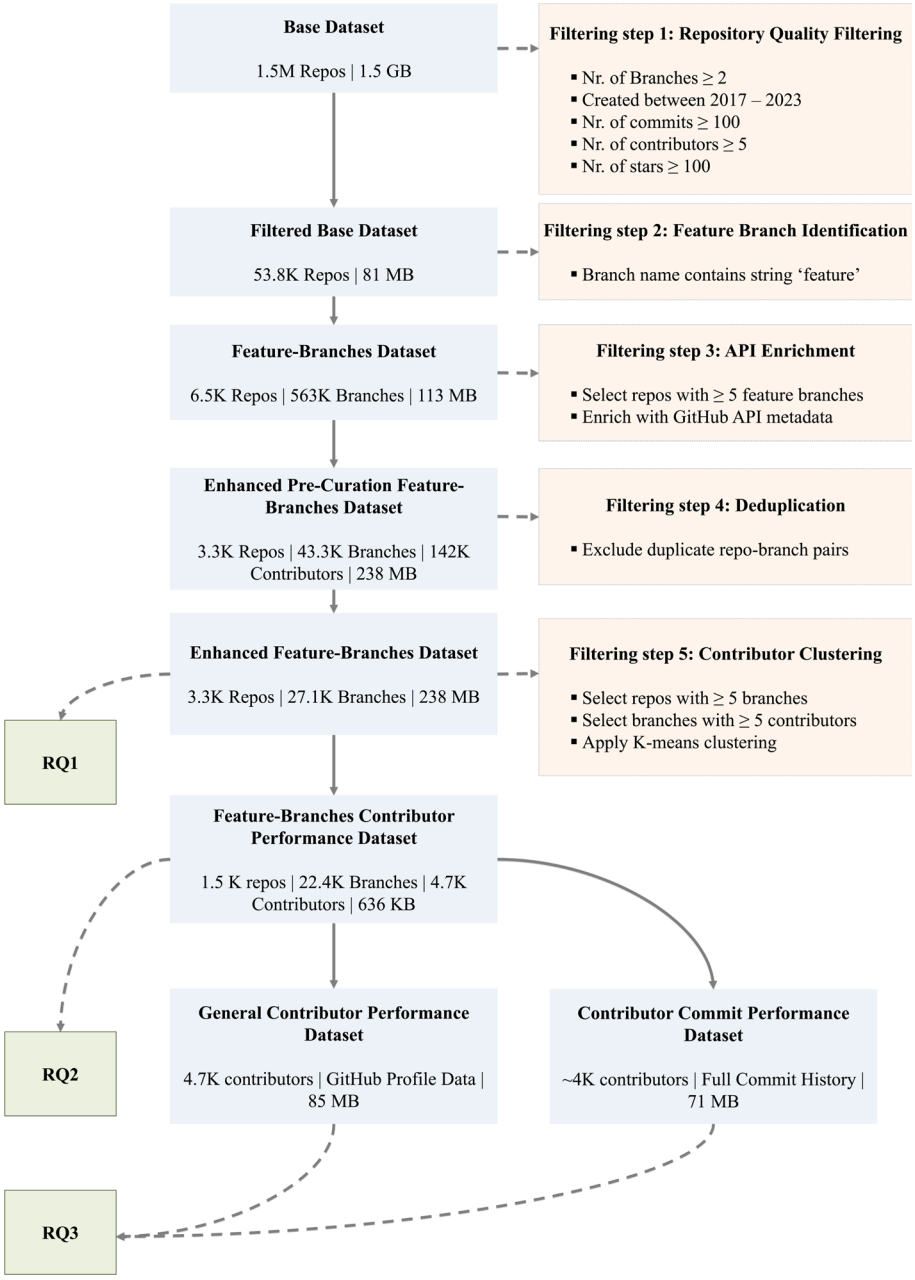


Fig. 1 Data collection and enrichment pipeline. The workflow filters the base dataset through multiple stages: filtering for quality criteria, extracting feature branches, enriching with GitHub API data, and generating three research question-specific outputs (RQ1: Branch Characteristics, RQ2: Contributor Distribution, RQ3: Contributor Profiles)

Table 1 Summary of datasets in the progressive collection pipeline

#	Dataset	Repos	Branches	Contributors	Size
1	Base	1,580,895			1.5 GB
2	Filtered Base	53,802	1,420,776		81 MB
3	Feature Branches	6,548	563,841[a]		113 MB
4	Enhanced Feature Branches	3,388	27,103[a]	142,268	238 MB
5	Feature-Branch Contributor Performance	1,550	22,489[a]	4,763	636 KB
6	General Contributor Performance			4,763	85 MB
7	Contributor Commit Performance			∼4,000	71 MB

[a] Feature branches only (not total number of branches)

(27,103 rows) used for RQ_1–RQ_2 analyses. Fourth, from the *Enhanced Feature-Branches dataset*, we derive the *Feature-Branches Contributor Performance dataset* by clustering contributor activity across branches, supporting RQ_2. Fifth, from the *Feature-Branches Contributor Performance dataset*, we aggregate contributor-level information to create the *General Contributor Performance dataset*, enabling RQ_3. Finally, we collect commit histories from contributors' own repositories to form a *Contributor Commit Performance dataset*, allowing cross-context validation of contributor activity.

All thresholds used in dataset construction (e.g., minimum contributors, minimum branches, percentile cutoffs) are aligned with the specific requirements of each research question and grounded in prior studies. Thresholds differ across datasets because repository-level filtering, branch-level analysis, and contributor-level clustering impose different statistical and computational constraints.

2.3 Base dataset

Our base dataset is obtained from the SEART GitHub Search platform, which provides indexed access to GitHub repositories and their associated metadata. The dataset was collected on August 8, 2024, without any initial filtering. It contains 1,580,895 repositories and is 1.5 GB in size and has a total of 35 columns. It is available in the replication package at *data/00-shared/base-dataset.csv*. SEART has been used in prior empirical software engineering studies (Cardoen et al. 2024) and provides efficient access to repository-level attributes, such as name, commit counts, contributor counts, stars, and branch counts.

2.4 Filtered base dataset

To focus on repositories exhibiting sustained, collaborative, and non-trivial development activity, we progressively applied the following filters to the base dataset.

- **Minimum branching activity.** We retained only repositories with at least two branches, ensuring the presence of at least one branch beyond the default branch. Repositories with a single branch rarely employ structured branching strategies and are unlikely to support feature-oriented workflows.
- **Temporal scope.** We restricted the dataset to repositories created between January 1, 2017 and December 31, 2023 to capture contemporary development practices and exclude legacy repositories predating modern GitHub-based workflows.
- **Minimum development activity.** Repositories with fewer than 100 commits were excluded. Commit count is commonly used as a proxy for sustained development activity and project maturity and has been employed in prior GitHub mining studies to filter inactive or non-engineered repositories (Dabic et al. 2021; Decan et al. 2022).
- **Collaborative development threshold.** We required repositories to have at least five contributors. This criterion filters out personal or experimental projects and ensures the presence of meaningful collaboration dynamics, which is essential for analyzing contributor behavior and coordination, and is consistent with contributor-based filtering used in prior OSS studies (Han et al. 2019). With fewer than five contributors, repositories represent solo or pair-developer projects unsuitable for studying team dynamics (RQ_2), where analyses of contribution inequality and performance clustering require sufficient sample sizes.
- **Community visibility.** Finally, we required repositories to have at least 100 stars. Stars are frequently used as a proxy for community adoption and project relevance and are commonly applied in large-scale OSS studies to curate high-quality repositories (Decan et al. 2022).

After applying these filters, the dataset was reduced to 53,802 repositories with 1,420,776 branches, forming the *Filtered Base dataset*. It is about 84 MB in size. It contains the same columns as the *Base dataset* and is available in the replication package at *data/00-shared/filtered-base-dataset.csv*.

2.5 Feature-branches dataset

From the *Filtered Base dataset* of 53,802 repositories (1,420,776 branches), we queried the GitHub API and retained branches whose names contained the substring "feature" (case-insensitive). Approximately 82% of these branches followed the "feature/" prefix convention (e.g., *feature/login-page*), consistent with established branching models such as Gitflow (Github 2024). We therefore labeled a branch as a feature branch if its name contained "feature" and retained only repositories with at least one such branch.

This resulted in 563,841 feature branches across 6,548 repositories, forming our *Feature-Branches dataset*. Because collection was performed in chunks without global deduplication, this file includes duplicate repository–branch entries (42,712 unique pairs). These records were used only as input to API-based enrichment.

The *Feature-Branches dataset* is about 119 MB in size and is available in the replication package at *data/01-feature-branches/feature-branches.csv* and contains a total of seven columns, such as RepositoryName, BranchName, LastCommitSHA, and CommitMessage. Repository names are also stored separately in *data/01-feature-branches/feature-branch-repository-names.csv* (150 KB).

2.6 Enhanced feature-branches dataset

To enable in-depth analysis of feature branch characteristics (RQ_1), we further enriched the *Feature-Branches dataset* with full commit histories and contributor information. Because this enrichment requires extensive GitHub API access, we applied filtering to balance analytical depth with computational tractability.

- **Outlier removal.** We first removed repositories with exceptionally high branch counts using the Interquartile Range (IQR) method, excluding repositories exceeding $Q_3 + 1.5 \times \text{IQR}$. This approach is standard in large-scale repository mining studies to eliminate extreme cases that could distort aggregate statistics (Borges et al. 2016). This step reduced the dataset to 5,742 repositories and 164,691 branches while preserving 87.7% of repositories, ensuring broad project coverage.
- **Minimum feature-branch count.** We excluded repositories with fewer than five feature branches to ensure sufficient feature-related activity for meaningful branch-level analysis. Repositories with too few feature branches may not reflect structured feature development practices and could represent experimental or transitional branching usage (Munaiah et al. 2017).
- **Maximum commit threshold.** We imposed an upper bound of 150,000 commits per repository to limit data volume and API request load. Based on preliminary analysis of the 5,742 candidate repositories, we estimated that without this threshold, processing would require approximately 863 days given GitHub's API rate limit of 5,000 requests per hour. With the threshold applied, actual collection is completed in about a month. The threshold was enforced dynamically during data collection by estimating expected commit counts as branches were processed, halting collection if the estimated total exceeded the limit. The average commit count per repository (41,690) remained well below the 150,000 threshold. Note that the 150,000-commit threshold was enforced dynamically during data collection rather than as a post-hoc filter, so a small number of repositories in the final dataset may exceed it; this threshold primarily served to limit API load during collection.
- **Runtime constraint.** Repositories for which retrieving branch commit histories exceeded 500 seconds were excluded to manage overall pipeline runtime and avoid API timeout issues. This constraint complements the commit threshold by preventing long-running API calls that could stall the collection process. After applying these collection criteria, the *Enhanced Feature-Branches pre-curation dataset (43,347 rows)* contains 3,388 repositories and 142,268 unique contributors. It has a total of eight columns, including RepositoryName, BranchName, FirstCommitDate, LastCommitDate, NumberOfCommits, NumberOfContributors, Contributors (JSON list), and CurrentBranchLifespan. It is about 250 MB in size and is available at *data/01-feature-branches/enhanced-feature-branches.csv*.
- **Repository–branch deduplication.** After removing duplicate repository–branch entries and sorting by `LastCommitDate`, we obtain the *Enhanced Feature-Branches dataset*. It is available at *data/03-deduplicated/enhanced-feature-branches-dedup.csv* and contains 27,103 distinct feature branches across 3,388 repositories.

RQ_1–RQ_2 branch-level analyses use the *Enhanced Feature-Branches dataset* as the analysis basis.

Deleted feature branches are not included, as GitHub does not expose their commit histories after deletion; this biases the dataset toward surviving and longer-lived branches.

2.7 Feature-branches contributor performance dataset

To analyze contributor behavior within feature branches (RQ_2), we derived the *Feature-Branches Contributor Performance dataset* from the *Enhanced Feature-Branches dataset*.

- **Minimum five branches.** We included repositories with at least five branch rows on the *Enhanced Feature-Branches dataset*.
- **Minimum contributor threshold.** We excluded branches with fewer than five contributors to ensure robust clustering and avoid unstable partitions. With fewer contributors, K-Means clustering produces trivial splits, e.g., with three contributors, a 2-1 partition lacks statistical reliability. The five-contributor threshold is consistent with repository-level filtering (Section 2.4) and ensures sufficient sample sizes for meaningful inequality analysis.
- **Activity measurement.** For each remaining branch, we computed the number of commits per contributor. We use commit count as the primary activity metric because it is consistently available across repositories, directly reflects incremental development effort, and has been widely used in prior OSS research to measure developer contribution and authorship (Bird et al. 2011; Avelino et al. 2016). Our goal is to characterize contribution distribution and inequality rather than assess code quality or functional impact, which would require additional code-level analysis beyond the scope of this study.
- **Performance classification.** We applied K-Means clustering with $k = 2$ to separate contributors into two groups based on commit activity level. The cluster with the higher mean commit count was labeled as *high-performing*, and the other as *low-performing*. We chose K-Means over alternatives (e.g., truck factor analysis Avelino et al. (2016)) because it classifies all contributors relative to each branch's local contribution distribution, rather than identifying only the top critical developers. This branch-specific classification enables analysis of contribution inequality patterns across the entire contributor pool (RQ_2). We acknowledge that binary classification ($k = 2$) simplifies the contributor spectrum; a ternary split ($k = 3$) could capture "medium" performers. However, we chose $k = 2$ for three reasons: (1) stability with moderately-sized teams (5–10 contributors), where $k = 3$ requires larger samples for reliable partitioning (Cohen and Consens 2018; 2) comparability with prior work using binary core/peripheral distinctions in OSS collaboration (Mockus et al. 2002; Crowston and Howison 2006); and (3) effectiveness at capturing contribution concentration, which is RQ_2's primary focus. During pilot analyses, we explored alternative cluster sizes (e.g., $k = 3$) but found that resulting partitions were unstable for typical feature-branch team sizes, reinforcing the choice of $k = 2$ for reliable branch-level classification.

Contributor labels are aggregated across branches to form the *Feature-Branches Contributor Performance dataset*, which records each contributor's performance classification and total commit activity across all branches in which they participated and is used to answer RQ_2.

The *Feature-Branches Contributor Performance Dataset* contains data about 4,763 contributors derived from 22,489 feature branches across 1,550 repositories, with a total of 10 columns, including Username, Name, Email, Contributor Performance, High Performance Score, and Low Performance Score. It is about 650 KB in size and is provided in the replication package at *data/02-contributors/feature-branch-contributor-performance-dataset.csv*.

2.8 General contributor performance dataset

To analyze broader contributor behavior patterns beyond individual feature branches (RQ_3), we take the same 4,763 contributors from the curated *Feature-Branches Contributor Performance dataset* and enrich each with their general GitHub profile data via the GitHub API–public repos, followers, stars, company, bio, recent activity, opened issues/PRs. This enables comparison of general GitHub activity between high- and low-performing contributors.

- **High-performer criteria.** Contributors are classified as *high-performing* if they satisfy two conditions: (1) they were identified as high performers in at least four feature branches, and (2) their total commit volume across all branches falls above the 95th percentile ($\geq$ 3,012 commits). The four-branch threshold ensures that classification is based on consistent performance rather than isolated high activity in a single branch, reducing classification noise. The 95th percentile cutoff identifies the top 5% most active contributors in the dataset, ensuring that only contributors with sustained, exceptional commit volumes are labeled as high performers.
- **Low-performer criteria.** Contributors are classified as *low-performing* if they were never identified as high performers in any branch but still fall within the top 10% of contributors by total commit volume. This criterion isolates contributors who consistently make modest contributions across many branches, distinguishing them from occasional or one-time contributors who might fall below the top 10%. Similar multi-category contributor classifications have been used in prior OSS studies (Mockus et al. 2002; Padhye et al. 2014).
- **Profile metadata collection.** For each classified contributor, we collected GitHub profile metadata to examine whether feature-branch performance correlates with broader activity patterns. Profile attributes include activity metrics (opened issues, pull requests), social engagement (followers, following counts), and repository ownership (public repositories, stars, forks). Prior work has shown that activity traces and social metrics serve as signals of developer engagement in GitHub (Marlow and Dabbish 2013), and that stars are widely used as indicators of repository popularity and community interest (Borges et al. 2016).

The resulting *General Contributor Performance dataset* is about 89 MB in size and contains a total of 18 columns, including Total Size of Repositories (KB), Recent Commit Activity, Opened Issues, and Opened Pull Requests. For (RQ_3), contributors appearing across

all three datasets are analysed jointly, and outliers are removed per metric using the IQR method prior to statistical testing. It is provided in the replication package at *data/02-contributors/general-contributor-performance-dataset.csv*.

2.9 Contributor commit performance dataset

To validate whether contributor activity within feature branches aligns with their broader coding behavior, we constructed the *Contributor Commit Performance dataset* by retrieving commit histories from each contributor's own public repositories.

- **Data collection period.** Using contributor usernames, we retrieved commit histories from each contributor's public repositories over a 24-month period (September 2022 to September 2024). This timeframe captures recent activity while providing sufficient temporal coverage to assess sustained contribution patterns.
- **Commit-level attributes.** For each commit, we recorded the timestamp, repository name, and repository owner (distinguishing contributions to the contributor's own repositories from contributions to others' repositories). Commit histories are aggregated per contributor to compute total commit volume and temporal distribution patterns outside the feature-branch context.

This dataset is about 74 MB in size and has a total of six columns, including Username, Name, Email, Contributor Performance, Commit History, and Total Commits. It is provided in the replication package at *data/02-contributors/contributor-commit-performance.csv*. This dataset enables cross-context comparison of contributor activity: do contributors who exhibit high performance in feature branches also show high commit activity in their own repositories? Such validation helps determine whether feature-branch performance reflects general coding productivity or context-specific factors (e.g., repository familiarity, feature complexity, collaboration dynamics).

Table 1 provides a consolidated overview of all the datasets and their key characteristics.

3 Data analysis and results

In this section, we report the empirical analyses conducted to address the three research questions. Using the curated datasets described in Section 2, we analyze (i) the characteristics and usage of feature branches (RQ_1), (ii) the distribution of contributions and collaboration patterns within feature branches (RQ_2), and (iii) the relationship between feature-branch performance and contributors' broader GitHub activity (RQ_3). The section focuses on descriptive statistics; interpretation of the results and their implications for practice and research are discussed in Section 4.

3.1 RQ1 how are feature branches used in open-source projects, and what are their key characteristics?

This subsection reports empirical characteristics of feature branches in open-source projects. Using the *Enhanced Feature-Branches dataset*, we analyze the number of feature

branches per repository, their lifespan, commit activity, contributor involvement, and temporal development intensity. Table 2 summarizes these measures. Unless stated otherwise, branch-level rows use the analysis sample of 23,807 branches from repositories with at least five feature-branch rows; the "Outliers Removed" column reports the share of observations excluded for that row's statistic (the exclusion rule differs by metric and is specified where each measure is discussed below). The first two rows are repository-level counts over all 3,388 repositories in the dataset—without IQR trimming and after 4×IQR removal of the most extreme repository counts (2.89% of repositories excluded)—as detailed in Section 3.1.1. Lifespan, commits per branch, average commits per day, and commits per contributor apply the filters described in Sections 3.1.2–3.1.7.

3.1.1 Number of feature branches per repository

The number of feature branches varies substantially across repositories. Based on the *Enhanced Feature-Branches dataset* (27,103 distinct feature branches across 3,388 repositories), the number of feature branches per repository is right-skewed. Without IQR-based removal on repository-level counts, the mean is 8.00 branches per repository and the median is 4; however, the distribution has a long upper tail—the 95th percentile is 27 branches per repository, 7.8% of repositories have at least 20 feature branches, and the maximum is 742—meaning, a subset of projects hosts many more feature branches than typical. After 4× IQR-based outlier removal on those counts, the mean is 5.76 and the median is 3 ($n = 3{,}290$ repositories; 2.89% of repositories with the most extreme counts excluded).

Key takeaway Most repositories maintain a modest number of feature branches (median 4), but the corpus is right-skewed: a minority of projects account for more feature branches (mean 8.00), including repositories with dozens or hundreds of feature branches. IQR-based trimming reports a lower typical level (mean 5.76; median 3) by excluding the most extreme repositories from that summary.

Table 2 Summary Statistics for Feature Branch Characteristics RQ_1

Metric	Mean	Median	Outliers Removed	N (branches)
Branches per Repository (no IQR)	8.00	4	–	–
Branches per Repository (IQR)	5.76	3	2.89%	–
Branch Lifespan (days)	916.7	784.8	13.39%	20,619
Commits per Branch	1,641.39	973	7.97%	21,909
Avg. Commits per Day	1.87	1.35	10.55%	21,192
Commits per Contributor per Branch	48.16	31.71	2.95%	23,104

3.1.2 Lifespan of feature branches

The feature branch lifespan is computed from the parsed `CurrentBranchLifespan` field (derived from first/last commit timestamps on the branch). On the *Enhanced Feature-Branches dataset* after the repository filter ($\geq$5 feature-branch rows per repository; $n = 23{,}807$ branches), we restrict lifespan descriptives to branches whose `LastCommitDate` precedes 2024-01-01. This step excludes 13.4% of branches ($n = 3{,}188$) that were still active at collection in August 2024. Among the remaining $n = 20{,}619$ branches, the median lifespan is 784.8 days (approximately 2 years and 1.8 months). Mean and median additionally omit branches with extreme parsed durations under an upper-bound 1.5$\times$IQR rule on hours ($\approx$2.45% of the 23,807-branch sample). Without the 2024-01-01 restriction, the median parsed lifespan on the same filtered cohort is 922.1 days; both estimates exceed two years.

The distribution indicates that many feature branches persist for extended periods, with typical lifespans measured in years rather than short-lived branch cycles.

Key takeaway Feature branches commonly persist for long periods, with typical lifespans measured in years rather than weeks or months.

3.1.3 Relationship between number of contributors and feature branch lifespan

Figure 2 shows the relationship between the number of contributors involved in a feature branch and its lifespan. On the *Enhanced Feature-Branches dataset* after the repository filter ($\geq$5 feature-branch rows per repository), lifespan is measured in days between the first and last commit timestamps. After bilateral 1.5$\times$IQR removal on contributor count and

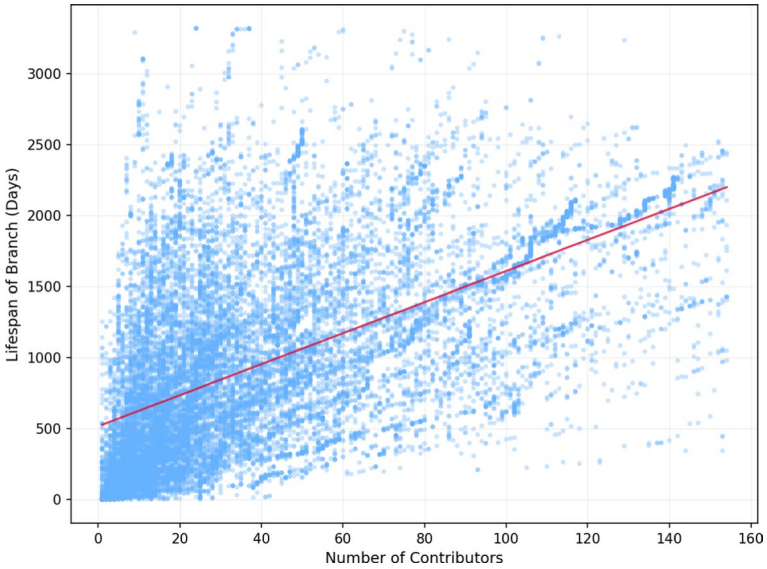


Fig. 2 Scatterplot of Number of Contributors vs. Lifespan of Branch (days from first to last commit; bilateral 1.5$\times$IQR on both variables; no 2024 last-commit cutoff)

lifespan, the correlation sample comprises $n = 22{,}216$ branches. The association is moderate and positive (Pearson $r = 0.579$, $p < 0.001$; $R^2 \approx 0.34$). Unlike the lifespan descriptives in Section 3.1.2, this analysis does not apply the 2024-01-01 `LastCommitDate` cutoff ($n = 20{,}619$). A total of 2,726 branches appear in Fig. 2 but not in Section 3.1.2 because their last commit occurred in 2024 or later. Feature branches with more contributors tend to remain active for longer periods.

Key takeaway Feature branches with more contributors tend to have longer lifespans, although the strength of this relationship is moderate.

3.1.4 Number of commits per feature branch

The total number of commits per feature branch shows substantial variation. After filtering extreme commit counts (3×IQR upper bound on `NumberOfCommits`), the mean number of commits per branch is 1,641.39 and the median is 973 ($n = 21{,}909$ feature branches on the *Enhanced Feature-Branches dataset* after the repository branch-count filter; Table 2).

Key takeaway Most feature branches accumulate a moderate number of commits, while a small number of branches account for very high commit volumes.

3.1.5 Relationship between number of contributors and commit volume

Figure 3 shows a moderate positive correlation between the number of contributors and the total number of commits per branch (Pearson $r = 0.571$, $p < 0.001$, $n = 20{,}522$ after bilateral 1.5×IQR on both variables), explaining approximately 33% of the variance ($R^2 \approx 0.33$). Without outlier trimming, the correlation is $r = 0.663$ ($n = 23{,}807$); the association is

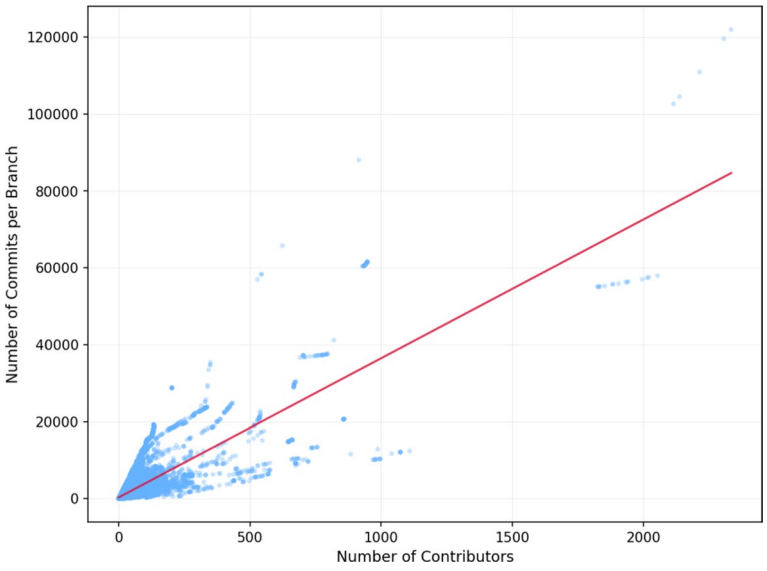


Fig. 3 Scatterplot of Number of Contributors vs. Number of Commits per Branch (bilateral 1.5×IQR on both variables; $n = 20{,}522$)

partly expected because larger teams can accumulate more commits by construction. Consistent with this association, feature branches with more contributors tend to accumulate more commits.

Key takeaway Feature branches with more contributors tend to accumulate more commits, though the association is moderate after standard outlier trimming ($r = 0.571$).

3.1.6 Average commits per day in feature branches

The average number of commits per day provides a measure of development intensity within feature branches. On the *Enhanced Feature-Branches dataset* after the repository branch-count filter, we compute each branch's average commits per day as `NumberOfCommits` divided by lifespan in days (from first to last commit timestamps). After IQR-based outlier removal on this rate (1.5×IQR), the mean is 1.87 commits per day and the median is 1.35 ($n = 21{,}192$ feature branches; 10.55% excluded; Table 2). The mean exceeds the median, indicating a right-skewed distribution in which a subset of branches exhibits higher daily commit intensity.

Key takeaway Daily commit activity in feature branches is generally low (median 1.35 commits per day), with a minority of more active branches raising the mean to 1.87.

3.1.7 Distribution of commits per contributor per branch

The number of commits per contributor per branch shows notable variation. On the *Enhanced Feature-Branches dataset* after the repository branch-count filter, we compute commits per contributor per branch as `NumberOfCommits` divided by `NumberOfContributors` for each branch. After upper-bound IQR outlier removal on this ratio ($Q_3 + 2.5 \times \text{IQR}$), the mean is 48.16 commits per contributor per branch and the median is 31.71 ($n = 23{,}104$ feature branches; 2.95% excluded; Table 2). The mean exceeds the median, indicating uneven contribution intensity across branches.

Key takeaway Commit contributions within feature branches are unevenly distributed across contributors.

Taken together, these results form the empirical basis for the findings related to RQ1, which are summarized in Table 5.

3.2 RQ2 how is development work distributed among contributors within feature branches, and what patterns of contribution emerge?

In this section, we analyze how development work is distributed within feature branches by examining contribution activity (e.g., commits per contributor), which allows us to identify recurring patterns of contribution. Using commit-based contribution measures, we analyze inequality at the branch and repository levels, the role of high-performing contributors, and the consistency of contributor performance across branches. Table 3 summarizes these measures. Gini coefficient rows and the 87.87% share with Gini > 0.6 use $n = 23{,}807$ branches

Table 3 Summary statistics for contributor distribution and performance RQ_2

Metric	Mean	Median	Min	Max
Gini Coefficient	0.742	0.773	0.00	0.970
Count of high performers	3.42	2.0	1	24
% of total contributors (high performers)	10.31%	8.0%	0.1%	85.7%
% commits by high performers	63.0%	63.24%	14.2%	99.7%
High-perf. consistency rate	–	90.6%	–	–
Low-perf. consistency rate	–	100.0%	–	–

after the $\geq$5-branches-per-repository filter on the deduplicated *Enhanced Feature-Branches dataset* (Section 2.6). High-performer count, contributor-share, commit-share, and consistency rows use $n = 22{,}489$ branches after contributor clustering (branches with $\geq$5 contributors per branch; Section 2.7).

3.2.1 Contribution inequality within feature branches

Contribution inequality within feature branches was measured using the Gini coefficient. Across branches, the mean Gini coefficient is 0.742, and the median is 0.773 (Table 3), with values ranging from 0.000 to 0.970. In total, 87.87% of branches ($n = 23{,}807$) exhibit a Gini coefficient above 0.6, indicating substantial contribution inequality.

A weak to moderate positive correlation is observed between the number of contributors and the Gini coefficient (Pearson $r = 0.353$, $p < 0.001$, $n = 21{,}730$ after IQR-based outlier removal), suggesting that inequality tends to increase with team size.

Key takeaway. Most feature branches exhibit high contribution inequality, with commit activity concentrated among a subset of contributors.

3.2.2 Proportion of high-performing contributors per branch

The distribution of the number of high-performing contributors per branch is right-skewed, with a mean of 3.42 and a median of 2.0 (Table 3). These results suggest that the majority of branches rely on a small core of contributors who drive development through higher volume or impact of commits.

When expressed as a percentage of total contributors per branch, high performers represent 10.31% on average (median 8.0%), ranging from 0.1% to 85.7%. This confirms that a small core group of contributors is responsible for most of the significant contributions in feature branches.

Key takeaway Feature branches typically involve a small number of high-performing contributors relative to the total number of contributors.

3.2.3 Percentage of commits by high-performing contributors

To assess the impact of high-performing contributors on feature branch development, we analyze the percentage of commits made by these contributors relative to the total commit activity in each branch. High-performing contributors account for a majority of commit activity: the mean percentage is 63.0%, and the median is 63.24% (Table 3).

These findings indicate that a small subset of contributors is responsible for most of the development and maintenance efforts in feature branches.

Key takeaway High-performing contributors account for the majority of commit activity within most feature branches.

3.2.4 Consistency of contributor performance across branches

Contributor performance consistency across branches was assessed using the Weighted Contributor High-/Low-Performance Rate. For each metric, the consistency rate is the percentage of contributors classified as high (low) performers who maintain that classification across all branches within their repository. Across repositories, the median commit-weighted high-performance rate is 90.6%, and the median low-performance rate is 100.0% (Table 3). This indicates that contributors identified as high performers tend to sustain their relative standing across branch contexts within a repository, while low performers show similarly stable patterns.

Key takeaway Contributor performance classifications tend to be consistent across branches within the same repository.

3.2.5 Impact of repository size on contributor performance

Repositories were grouped by the number of feature branches into three size categories (at or below the 25th percentile, between the 25th and 90th percentiles, and above the 90th percentile of branch counts per repository among repos with $\geq$5 clustered branches). Across these categories, high-performing contributors tend to participate across a larger fraction of branches in smaller repositories, whereas their branch-participation rate appears lower in repositories with many feature branches.

High-performing contributors also appear more selective in larger repositories, while low-performing contributors remain comparatively stable in lower relative-contribution roles.

Key takeaway High-performing contributors may become more selective as repository branch count increases, while low-performing contributors remain consistently low across repository sizes.

3.2.6 Weighted analysis of contributor performance

The Weighted High-/Low-Performance Rate reflects the proportion of commits made by contributors while classified as high or low performers across all feature branches. Among high performers, weighted contribution shares vary widely: some maintain high shares across branches, while others fluctuate more.

In contrast, weighted low-performance patterns are more concentrated, indicating comparatively stable lower relative contribution levels across branches.

Table 4 GitHub profile metrics: High vs. Low Performing Contributors RQ_3

GitHub Profile Metric	High Perf. (Mean)	Low Perf. (Mean)	Cohen's d	p-value
Opened Issues	458.79	208.66	0.90	<0.001
Opened Pull Requests	363.50	147.16	0.90	<0.001
Followers	47.10	18.94	0.79	<0.001
Public Repositories	30.59	22.65	0.33	<0.001
Total Stars	37.80	9.81	0.65	<0.001
Total Forks	11.37	3.60	0.64	<0.001
Total Repo Size (KB)	854,883	649,236	0.24	<0.001

Key takeaway Low contribution levels are more consistent across branches than high contribution levels.

Taken together, these results form the empirical basis for the findings related to RQ2, which are summarized in Table 5.

3.3 RQ3 can contributors' broader GitHub profile metrics help predict or explain their feature-branch performance?

This subsection compares contributors classified as high- and low-performing in feature branches with respect to selected GitHub profile metrics. The analysis is based on the *General Contributor Performance dataset* and examines differences in activity, visibility, and repository-related measures. All group comparisons were conducted using Mann–Whitney U tests, with effect sizes reported using Cohen's *d*. Rank-biserial correlations for the two activity metrics with the largest between-group differences are reported below. Summary statistics are shown in Table 4 (Figs. 4 and 5).

3.3.1 Issue and pull request activity

High-performing contributors exhibit higher levels of issue and pull request activity than low-performing contributors. For opened issues, high performers show a mean of 458.79

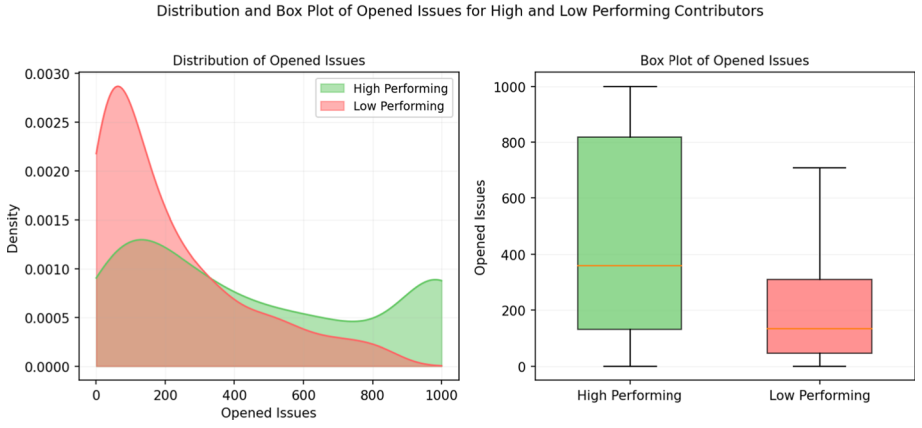


Fig. 4 Distribution and box plot of opened issues for high and low performing contributors

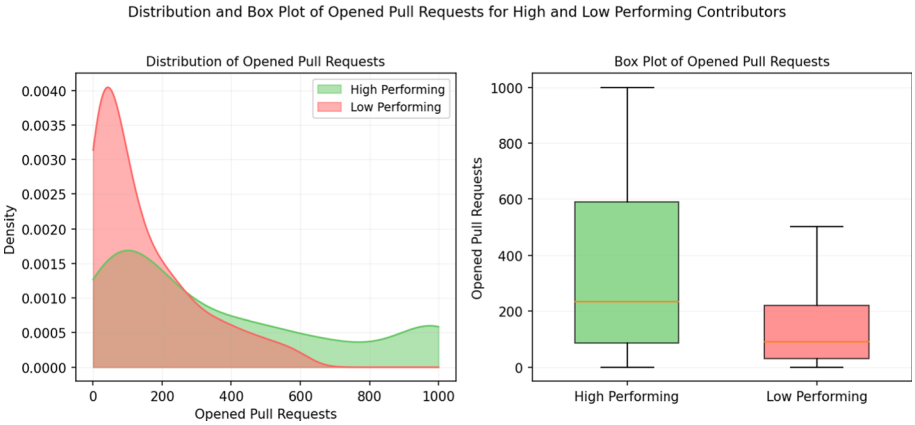


Fig. 5 Distribution and box plot of opened pull requests for high and low performing contributors

compared to a mean of 208.66 for low performers (Table 4), with median values of 359 and 133, respectively. All comparisons statistically significant at $p < 0.001$ (Mann–Whitney U tests). Cohen's *d*: small (0.2–0.5), medium (0.5–0.8), large (> 0.8). Rank-biserial correlations for opened issues and pull requests are 0.41 and 0.38, respectively. High performers: $n = 1,432$; Low performers: $n = 2,568$ (before outlier removal; per-metric 'n' varies after IQR-based outlier removal). For opened pull requests, high performers have a mean of 363.50 compared to a mean of 147.16 for low performers, with median values of 234 and 92, respectively.

Both differences are statistically significant ($p < 0.001$) and associated with large effect sizes (Cohen's $d = 0.90$ for issues, $d = 0.90$ for pull requests).

Key takeaway High-performing contributors open substantially more issues and pull requests than low-performing contributors.

3.3.2 Social visibility and repository ownership

High-performing contributors tend to have greater social visibility on GitHub. They have a higher number of followers, with a mean of 47.10 compared to 18.94 for low performers (Table 4), with median values of 27 and 10, respectively. This difference is statistically significant ($p < 0.001$) with a medium-to-large effect size (Cohen's $d = 0.79$).

High performers also own more public repositories, with a mean of 30.59 compared to 22.65 for low performers (median values of 22 and 15, respectively), and show higher aggregate counts of stars (mean 37.80 vs. 9.81, Cohen's $d = 0.65$) and forks (mean 11.37 vs. 3.60, Cohen's $d = 0.64$) across their repositories.

Key takeaway High-performing contributors generally have higher visibility and ownership metrics than low-performing contributors.

3.3.3 Repository size

High-performing contributors tend to maintain larger repositories than low-performing contributors. The mean total repository size for high performers is 854,883 KB, compared to 649,236 KB for low performers (Table 4), with median values of 467,342 KB and 328,319 KB, respectively. Although the difference is statistically significant ($p < 0.001$), the effect size is small (Cohen's $d = 0.24$).

Key takeaway Differences in repository size between high- and low-performing contributors are present but comparatively limited.

Taken together, these results provide the empirical basis for the findings related to RQ3, which are summarized in Table 5.

Table 5 Summary of key findings from feature-branch analysis

Finding	RQ	Key Result	Implication
Fin_1	RQ_1	Typical feature branches accumulate on the order of 1,000 commits over a median lifespan of approximately two years among branches inactive before January 2024, corresponding to low average daily commit activity.	Indicates that feature-branch development often progresses gradually over long periods rather than through short bursts of activity.
Fin_2	RQ_1	An increase in the number of contributors is associated with higher commit volume and longer feature-branch lifespans.	Suggests that branches with more contributors tend to remain active longer and accumulate more commits, which may increase coordination requirements as teams grow.
Fin_3	RQ_1	Across repositories and branches, mean values for branch counts, commit volume, and contributor activity substantially exceed median values.	Shows that a small number of large branches and highly active contributors account for a disproportionate share of development activity.
Fin_4	RQ_1	Many repositories maintain several long-lived feature branches concurrently.	Points to sustained parallel development and increased demands on integration and coordination mechanisms.
Fin_5	RQ_2	On average per branch, 10.31% of contributors account for about 63.0% of commits on that branch, indicating high contribution inequality with a mean Gini coefficient of 0.742.	Highlights strong dependence on a small subset of contributors for the branch-level development.
Fin_6	RQ_2	High-performing contributors participate across a larger fraction of branches in smaller repositories and more selectively in larger ones.	Suggests that contribution patterns differ with project scale and may require different coordination approaches.
Fin_7	RQ_3	Contributors classified as high performing in feature branches tend to have higher levels of activity and visibility on GitHub, including substantially more issues opened, pull requests, followers, and larger repository portfolios.	Indicates that broader GitHub profile metrics are associated with feature-branch performance and may serve as signals for identifying strong contributors.

3.4 Summary of key findings

Table 5 consolidates the key findings derived from the results of RQ1–RQ3. Each finding synthesizes multiple quantitative observations reported in Section 3 and is interpreted in more detail in Section 4.

4 Discussion

This study examined how feature branches are used in OSS projects and how development work is distributed within them, with a particular focus on branch lifespan, contribution inequality, and contributor performance profiles. In this section, we interpret the findings in relation to the research questions and discuss their implications for practice and future research.

4.1 Implications for feature-branch management

In response to RQ1, the findings highlight how feature branches are used over time and what this means for branch management practices. The prevalence of long-lived feature branches indicates that, in practice, feature branches are often used to support development work over extended periods rather than being short-lived branches meant for frequent merging. While this may reflect the complexity of certain features or coordination challenges in OSS projects, it may also increase integration risk in the sense discussed in the CI/CD literature. CI/CD and agile guidance often recommends short-lived branches (roughly one to four months), moderate daily commit rates, and small contributor teams, on the grounds that such branches may reduce integration risk (Agrawal and Chari 2007). In contrast, the RQ_1 results above describe a markedly different profile: long-lived branches, large contributor teams, and low daily commit intensity rather than short sprint-length cycles. We did not measure merge frequency, merge conflicts, or branch–main divergence; integration-risk statements below follow general CI/CD reasoning rather than outcomes reported in this study.

From a practical perspective, these results suggest that tracking how long branches stay open is relevant for OSS management and maintenance. Rather than imposing strict time limits, maintainers may consider simple signals, such as branch age, how long a branch has been inactive, or long periods without merging into the main branch, as reasons to take a closer look. Practices, such as more frequent merging or regular reviews of long-running branches, could help reduce integration problems while preserving flexibility in development.

Our data suggest several practical signals where maintainers might consider assessing a branch's status:

- **Branch Open for >6 Months:** At this point, maintainers might ask: Is this feature still needed? Are there blockers preventing progress?
- **No Activity for 2–3 Months:** When a branch experiences extended inactivity, it could indicate abandonment or reduced feature priority. A brief assessment of whether the work remains relevant might help prevent branches from becoming stale.
- **One Person Doing Most of the Work:** When a single contributor accounts for a large

share of commits on a branch (in our sample, the median top-contributor commit share is 34.6%; 8.3% of branches exceed 85%), it may create concentration risk. In such cases, involving others to review and contribute to the code could be important for knowledge distribution.

Branches that meet the following conditions may warrant closer attention:

- **Branch Open for >12 Months:** At this threshold, integration risk could become substantial. The branch code may have diverged significantly from the main branch, potentially making merging complex.
- **Single-Contributor Branches:** When all work is contributed by one person, involving others could help distribute knowledge and reduce concentration risk.

It is important to note that each project may have different characteristics and practices. A small project with few contributors may operate effectively with longer-lived branches, whereas a large project might benefit from faster integration cycles.

The thresholds above (e.g., >6 months open, high top-contributor commit share) are starting points, not universal rules. Contribution concentration may also vary with repository scale. A single-contributor branch may be unremarkable in a small project but warrants attention in a larger one. Teams can calibrate to their own context by computing contributor concentration (e.g., top-contributor commit share or Gini coefficient) over their recently merged branches and flagging branches that fall outside their historical norm. These checks are straightforward to automate: branch age and inactivity are queryable via the GitHub API, and GitHub Actions can post reminders or apply labels when a branch crosses a threshold.

Beyond branch age, rolling commit intensity could also offer a reasonable signal in the context of open-source projects, where contribution is often asynchronous and volunteer-driven. Since daily commit activity is generally low and irregular (median 1.35 commits/day; Section 3.1.6), a persistent drop—for example, fewer than one commit per week over a four-week window—may, as a practical heuristic rather than a study-derived threshold, indicate a stalled branch earlier than age alone. This metric is computable directly from commit timestamps and could be tracked automatically alongside branch age checks.

Summary of practical implications

- Long-lived feature branches can increase integration risks and may warrant monitoring.
- Branch age and periods of inactivity may serve as signals that could prompt further review.
- More frequent merging and regular branch reviews might help manage long-running branches.

4.2 Implications for contribution inequality and team risk

In response to RQ2, the findings show how development work is distributed within feature branches and what risks may arise from unequal contribution patterns. The strong concentration of contributions within feature branches highlights risks related to reduced knowledge distribution and dependence on a small number of contributors. When branch-level

work is largely carried out by a few individuals, the continuity of development may be affected if these contributors reduce their involvement or leave the project.

At the same time, unequal contribution patterns are not necessarily negative. They may reflect deliberate task allocation or the involvement of contributors with specific expertise. Potential issues arise primarily when such concentration is not visible or not considered in project coordination. The results indicate that examining how contributions are distributed within feature branches can help identify situations where knowledge sharing may be limited.

For practitioners, this suggests that contribution inequality should be interpreted as a characteristic of branch-level work that warrants attention. Practices such as code review involving multiple contributors, shared responsibility for important branches, or improved documentation around feature-related decisions may help reduce dependency on individual contributors while maintaining the advantages of specialization.

Summary of practical implications

- Contribution inequality can indicate dependency risks when work is concentrated among a few contributors.
- Such patterns warrant monitoring and potential attention.
- Code review involving multiple contributors and improved documentation might help reduce knowledge concentration.

4.3 Implications for identifying and supporting contributors

In response to RQ3, the findings illustrate how contributor performance varies across feature branches and how such patterns may inform task allocation. The analysis shows that contributors classified as high performing are often responsible for a large share of the work within feature branches. In smaller repositories, high-performing contributors tend to participate across a larger fraction of branches; in larger repositories, their work is more concentrated on fewer branches.

These observations suggest two practical directions for OSS maintainers. First, contributors who repeatedly account for a large share of commits across multiple branches within a repository are often core developers who carry substantial implementation knowledge. Recognizing this may help maintainers prioritize retention, for instance, by distributing review responsibilities more deliberately. Second, when broadening participation on a branch, maintainers may find GitHub profile metrics—such as opened issues and pull requests—useful as supplementary signals of platform experience, interpreted together with in-project contribution history.

At the same time, these observations need to be interpreted carefully. The notion of performance used in this study is based on commit activity and does not capture other forms of contribution, such as design discussions, code reviews, or documentation. Profile activity counts may reflect cumulative engagement rather than causally preceding branch performance. Any practical use of these metrics should account for their limited scope and the potential for incomplete assessment when interpreting contributor roles.

Summary of practical implications Summary of practical implications

- Understanding consistent contribution patterns may help identify key contributors and their roles.
- Contribution metrics should be used cautiously and in conjunction with other contextual information.

4.4 Implications for research

From a research perspective, the results indicate that feature branches should be examined not only in terms of their technical properties but also with respect to how they are used to structure and distribute development work among contributors. The observed patterns suggest that branch management practices are closely related to factors such as team size, project organization, and contributor roles.

An important avenue for future research is to refine and extend the feature branch identification approach. Our naming heuristics may miss branches implementing feature-related work without following standard naming conventions. Understanding the prevalence and characteristics of such branches through qualitative investigation or manual sampling could provide insights into whether our findings generalize broadly or are specific to projects that follow standard naming conventions. Since branch naming conventions may vary across projects, communities, and development contexts, validation work would help the community assess the scope and limitations of heuristic-based feature branch analyses.

Beyond validation efforts, future studies could explore alternative ways of characterizing contributor involvement by incorporating additional forms of activity, such as review participation or issue handling, or by analyzing how branch-related practices evolve over time. In addition, comparing open-source projects with industrial projects may help distinguish patterns that are specific to OSS settings from those that also apply in other development contexts.

5 Threats to validity

Although we employ systematic methods and large-scale data, we acknowledge potential limitations in construct, internal, and external validity that may affect the generalizability and interpretation of our findings.

5.1 Construct validity

Feature branches were identified using a name-based heuristic (case-insensitive substring match for “feature”), based on established naming conventions such as “feature/”. As reported in Section 2.5, a substantial fraction of branches follow the canonical “feature/” prefix pattern. These statistics indicate that our heuristic captures a dominant and semantically specific naming convention, making accidental matches unlikely and suggesting high precision.

However, we did not perform a systematic manual audit to estimate false-positive or false-negative rates. In particular, recall remains unknown: branches implementing feature-related work without using common naming conventions may not be included. Con-

versely, some branches containing the term "feature" may not strictly correspond to feature development.

This limitation primarily affects conclusions about absolute prevalence and counts of feature branches (RQ_1), where missed branches may lead to conservative underestimation. RQ_2 and RQ_3, which focus on contribution distributions and clustering patterns within the identified sample, are less sensitive to occasional misclassification. Unless missed branches systematically differ in their contribution dynamics, the observed patterns are expected to remain stable.

Misclassification of what counts as a feature branch is unlikely to distort aggregate distributional patterns when errors are sporadic and the analysis unit is distinct repository–branch pairs ($n = 27{,}103$ on the *Enhanced Feature-Branches dataset*). Using pre-curation row counts (43,347) as branch counts would overstate sample size because chunked enrichment can duplicate pairs; reported branch statistics therefore use the deduplicated branch-level dataset throughout RQ_1–RQ_2. The large sample size ($n = 27{,}103$ distinct feature branches across 3,388 repositories) further reduces the likelihood that random misclassification substantially alters overall patterns.

Contributor performance was operationalized using commit-based clustering into high- and low-performing groups. We operationalize the distribution of development work through contribution activity, which limits conclusions to the contribution structure and does not directly measure collaboration or coordination mechanisms. While this approach allows consistent comparison across branches, it simplifies contribution patterns and does not represent intermediate or non-code contributions. This limitation affects the interpretation of contribution inequality and contributor performance results (RQ2, RQ3). Because K-Means assigns labels relative to each branch's local commit distribution, "high-performing" and "low-performing" denote within-branch contribution share rather than organization-wide productivity or code quality—supporting RQ_2's inequality focus but limiting extension to other work contexts (Section 2.7, 2.8). RQ_3 profile comparisons further restrict to contributors meeting sustained high/low criteria across multiple branches ($n = 4{,}000$ after cohort selection: 1,432 high-performing, 2,568 low-performing), meaning those findings characterize a defined performance subset rather than GitHub developers in general.

5.2 Internal validity

The analysis relies on several repository-level and branch-level filtering thresholds. These thresholds were selected based on established practice in large-scale OSS studies to exclude inactive or non-engineered projects, but alternative choices could lead to different dataset compositions and influence results across all research questions (RQ1–RQ3). The primary RQ_1–RQ_2 subsample—repositories with at least five feature-branch rows and branches with at least five contributors for clustering—excludes lightly active projects where branch-level patterns may be unstable (Section 2.7). This bounds prevalence and inequality estimates to repositories with established feature-branch practice rather than the full enhanced cohort; the replication package documents exact counts and supports re-analysis under alternative cutoffs.

Chunked GitHub API enrichment can yield multiple rows for the same repository–branch pair; all branch-level analyses therefore use one row per pair, retaining the snapshot with the latest `LastCommitDate` ($n = 27{,}103$; Section 2.6).

Outlier removal using the interquartile range was applied to limit the influence of extreme values. This step improves robustness for large-scale analysis, but may exclude valid projects or branches with unusually high activity. As a result, findings related to branch lifespan and contribution distribution should be interpreted as representative of typical projects rather than extreme cases (RQ1, RQ2).

Branch lifespan descriptives additionally exclude branches whose last commit occurred on or after 2024-01-01 (13.4% of the RQ_1–RQ_2 filtered sample), because these branches were still active at collection (August 2024) and are right-censored rather than statistical outliers; reported medians therefore understate longevity relative to the full filtered cohort.

Binary high/low labels from K-Means ($k = 2$) depend on cluster initialization and library version. We document the clustering configuration and pin `scikit-learn`==1.5.1 in the replication package for reproducibility; pilot analyses explored alternative k values with unstable partitions for typical branch team sizes (Section 2.7). Inequality patterns are therefore conditional on this operationalization, though it is applied uniformly across all branches in the clustering sample.

5.3 External validity

The study excludes feature branches that were deleted prior to data collection. Since short-lived branches are more likely to be deleted, our lifespan and prevalence estimates for RQ1 may be biased toward longer-lived branches and should be interpreted accordingly.

Finally, the analysis focuses on open-source projects hosted on GitHub and relies on publicly available data. While this enables large-scale analysis, the results may not directly generalize to industrial settings or to development conducted primarily in private repositories, particularly with respect to contributor roles and performance patterns (RQ2, RQ3).

6 Related work

6.1 Feature definitions and conceptual foundations

Many existing studies have focused on formalizing the definition of "feature" across different contexts, providing structured categorizations and guidelines for consistent usage (Classen et al. 2008; Apel and Kästner 2009; Banerjee et al. 2012; Berger 2015; Kala and Nandhini 2018; Molenaar et al. 2022). They define features from perspectives such as requirements engineering, software product lines, and information science. However, these studies primarily address conceptual clarity rather than empirically analyzing how features are developed and managed in practice.

6.2 Developer activity and contribution patterns

Research on developer activity in OSS has examined various aspects of contribution patterns, from individual coding behaviors to project-level success factors.

Chatziasimidis et al. analyze contributor activity to identify project success factors, focusing on metrics such as user activity and collaborator count (Chatziasimidis and Stame-

los 2015). In contrast, our research examines contributor behavior specifically within the context of feature branch development, providing a more fine-grained view of how developers collaborate on individual features.

Onoue et al. explore the diversity of developer behaviors across coding, commenting, and issue handling, though their study is limited to two OSS projects (Onoue et al. 2013). Our work extends this line of inquiry by analyzing developer activity across 3,388 repositories, focusing specifically on feature branch contexts.

6.3 Communication and collaboration in OSS development

Ortu et al. provide insights into contributor roles and communication patterns through comment analysis in OSS projects (Ortu 2018), focusing on linguistic and sentiment aspects of developer interactions. Guzman et al. analyzed 60,425 commit comments from 90 projects to understand developer sentiments (Guzman et al. 2014). While these studies examine communication channels (comments, issues), our work focuses on commit-based activity patterns within feature branches, though we acknowledge that asynchronous communication through pull requests and issues plays a crucial role in coordination (as discussed in Section 4.1).

6.4 Repository mining and empirical software engineering

Beyond contributor-focused studies, broader repository mining research has examined various aspects of OSS development. Borges et al. studied factors that impact the popularity of GitHub repositories (Borges et al. 2016). Qi et al. mined GitHub to collect diverse real-world effort data for effort estimation (Qi 2017). Rahman et al. compare successful and unsuccessful pull requests across 78 GitHub projects (Rahman and Roy 2014), examining factors that influence PR acceptance rates. While these studies provide valuable insights into general repository characteristics and pull request dynamics, our work specifically targets feature branch lifecycles and contributor performance within these branches.

6.5 Contribution inequality in open-source projects

Our finding that, on average per feature branch, roughly 10% of contributors account for over 60% of that branch's commits aligns with broader research on contribution inequality in OSS (global contributor–branch assignment shares are lower; see Section 3.2.2). Xia et al. analyzed 860 million GitHub events and found that the top 20% of contributors accounted for 89% of all activity (Xia et al. 2022). Padhye et al. confirmed similar patterns across different programming languages, showing that a small core group dominates commit activity (Padhye et al. 2014). This Pareto-like distribution has important implications for project sustainability, knowledge management, and community health. Our work extends this line of research by examining contribution inequality specifically within feature branches, providing insights into how these patterns manifest during feature development rather than across entire repositories.

6.6 Branching strategies and version control practices

Kovalenko et al. evaluated the importance of correctly handling branches when calculating file modification histories (Kovalenko et al. 2018), highlighting that branch-aware analysis is essential for accurate software evolution studies. Our work builds on this insight by treating feature branches as first-class analytical units rather than simply merging all commits into a single timeline. While existing research on branching strategies has primarily focused on best practices and workflow patterns, empirical evidence about how these strategies are actually used in practice remains limited. Our study addresses this gap by providing large-scale empirical data on feature branch lifespans, commit patterns, and contributor participation.

6.7 Continuous integration and agile development practices

Our findings regarding long-lived feature branches (median lifespan exceeding two years among branches inactive before January 2024, and ≈922 days on the full filtered cohort) contrast sharply with recommendations from the continuous integration and agile development literature, which advocate for short-lived branches that are frequently merged (Apel and Kästner 2009; Rahman et al. 2016). This disconnect between prescribed best practices and actual developer behavior has been noted in other contexts as well, suggesting that real-world constraints—such as coordination overhead, code review bottlenecks, and feature complexity—may prevent teams from adhering to ideal workflows. Our work provides empirical evidence of this gap and suggests that research on branching strategies should account for the practical realities of OSS development, where volunteer contributors, asynchronous collaboration, and fluctuating project priorities are common.

7 Conclusion and next steps

In this study, we analyzed 27,103 feature branches from 3,388 open-source repositories and found that they often remain active for over two years, far exceeding typical agile sprint durations. A small cohort of high-performing contributors, representing approximately 10.31% of contributors per branch on average, produces about 63.0% of commits within feature branches and maintains consistent top performance across multiple branches. This heavy reliance on a narrow group of high performers poses challenges such as knowledge silos, diminished bus factor, and potential burnout risks.

Our results have several practical implications for feature-branch management. We recommend monitoring branch age and inactivity as signals for review, with particular attention to branches open for over 6 months or experiencing 2–3 months of inactivity. For contribution inequality, we suggest employing code review practices involving multiple contributors and improving documentation to mitigate knowledge concentration risks. For contributor identification and support, we recommend identifying key contributors through contribution pattern analysis.

In future work, we intend to expand our scope beyond the *contributor dimension* by investigating how code-level aspects (e.g., complexity and refactoring activity) evolve within long-lived feature branches, as well as exploring requirements-level analyses to assess whether developers maintain well-defined feature boundaries in practice. Addition-

ally, we aim to examine feature-related GitHub issues to see how contributors discuss, track, and resolve feature implementation tasks. Finally, we will assess whether similar patterns appear in private or closed-source repositories, where structured team dynamics and corporate policies may shape collaboration and performance outcomes differently.

Author Contributions Dr. Nitish Patkar and Mr. Aimen Fahmi contributed equally to this work. Dr. Nitish Patkar, Prof. Dr. Timo Kehrer, and Mr. Aimen Fahmi contributed to the conception and design of the study. Data collection and analysis were performed by Dr. Nitish Patkar and Mr. Aimen Fahmi. The first draft of the manuscript was written by Dr. Nitish Patkar. Prof. Dr. Timo Kehrer and Prof. Dr. Norbert Seyff provided reviews and feedback on the manuscript. All authors read and approved the final manuscript.

Funding This work has been supported by the Swiss National Science Foundation (SNSF) under grants 219719 and 222903.

Data Availability The datasets generated, all scripts used for data collection and analysis are available in the supplementary material: https://figshare.com/s/7badb8e4a67380b2519e.

Declarations

Ethical Approval Not applicable. This study used only publicly available open-source software data and did not involve human subjects or personal data.

Informed Consent Not applicable. This study used only publicly available open-source software data and did not involve human subjects or personal data.

Conflicts of Interest The authors declare that they have no competing interests.

Clinical Trial Number Not applicable.

References

Agrawal M, Chari K (2007) Software effort, quality, and cycle time: a study of cmm level 5 projects. IEEE Trans Softw Eng 33:145–156

Apel S, Kästner C (2009) An overview of feature-oriented software development. J Object Technol 8:49–84

Avelino G, Passos L, Hora A, Valente MT (2016) A novel approach for estimating truck factors. IEEE, pp 1–10

Banerjee M, Roy SR, Kumar C (2012) Feature oriented programming: a step towards flexible composition of modular programming. IEEE, pp 369–373

Berger T (2015) What is a feature? A qualitative study of features in industrial software product lines. pp 16–25

Bird C, Nagappan N, Murphy B, Gall H, Devanbu P (2011) Don't touch my code! examining the effects of ownership on software quality. pp 4–14

Borges H, Hora A, Valente MT (2016) Understanding the factors that impact the popularity of github repositories. IEEE, pp 334–344

Cardoen G, Mens T, Decan A (2024) A dataset of github actions workflow histories. pp 677–681

Chatziasimidis F, Stamelos I (2015) Data collection and analysis of github repositories and users. IEEE, pp 1–6

Classen A, Heymans P, Schobbens P-Y (2008) What's in a feature: a requirements engineering perspective. Springer, pp 16–30

Cohen E, Consens MP (2018) Large-scale analysis of the co-commit patterns of the active developers in github's top repositories. pp 426–436

Crowston K, Howison J (2006) Hierarchy and centralization in free and open source software team communications. Knowl Technol Policy 18:65–85

Dabic O, Aghajani E, Bavota G (2021) Sampling projects in github for msr studies. IEEE, pp 560–564

Decan A, Mens T, Mazrae PR, Golzadeh M (2022) On the use of github actions in software development repositories. IEEE, pp 235–245
GitHub (2024) Inc. About branches: proposing changes to your work with pull requests. https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-branches. Accessed: Feb 4, 2026
Guzman E, Azócar D, Li Y (2014) Sentiment analysis of commit comments in github: an empirical study. pp 352–355
Han J, Deng S, Xia X, Wang D, Yin J (2019) Characterization and prediction of popular projects on github. IEEE, vol 1, pp 21–26
Kala K, Nandhini M (2018) A systematic literature review on feature oriented software development paradigm. In: International journal of engineering and technology (IJET)
Kalliamvakou E et al (2016) An in-depth study of the promises and perils of mining github. Empir Softw Eng 21:2035–2071
Kovalenko V, Palomba F, Bacchelli A (2018) Mining file histories: should we consider branches? pp 202–213
Marlow J, Dabbish L (2013) Activity traces and signals in software developer recruitment and hiring, CSCW '13, pp 145–156
Mockus A, Fielding RT, Herbsleb JD (2002) Two case studies of open source software development: apache and mozilla. ACM Trans Softw Eng Methodol (TOSEM) 11:309–346
Molenaar S, Steenvoorden E, van den Berg N, Dalpiaz F, Brinkkemper S (2022) Defining key concepts in information science research: The adoption of the definition of feature. Springer, pp 442–457
Munaiah N, Kroh S, Cabrey C, Nagappan M (2017) Curating github for engineered software projects. Empir Softw Eng 22:3219–3253
Onoue S, Hata H, Matsumoto K-I (2013) A study of the characteristics of developers' activities in github. IEEE, vol 2, pp 7–12
Ortu M (2018) Mining communication patterns in software development: A github analysis. pp 70–79
Padhye R, Mani S, Sinha VS (2014) A study of external community contribution to open-source projects on github. pp 332–335
Patkar N, Fahmi A, Kehrer T, Seyff N (2025) What is a feature, really? Toward a unified understanding across se disciplines. IEEE, pp 41–45
Qi F (2017) Software effort estimation based on open source projects: case study of github. Elsevier, vol 92, pp 145–157
Rahman MM, Roy CK (2014) An insight into the pull requests of github. pp 364–367
Rahman MT, Querel L-P, Rigby PC, Adams B (2016) Feature toggles: practitioner practices and a case study. pp 201–211
Xia X, Weng Z, Wang W, Zhao S (2022) Exploring activity and contributors on github: who, what, when, and where. IEEE, pp 11–20

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.



Dr. Nitish Patkar Nitish Patkar is a Senior Researcher at the University of Applied Sciences and Arts Northwestern Switzerland (FHNW). His research interests span several subfields of software engineering, including requirements engineering, agile methodologies, empirical software engineering, and software sustainability. He is an associate member of the International Requirements Engineering Board (IREB) and regularly serves as a program committee member and organizer for international conferences such as REFSQ, RE, ICT4S, and XP.



Mr. Aimen Fahmi Aimen Fahmi is a researcher in software engineering with interests in empirical software engineering, open-source software development, and feature engineering. His work investigates how software features are understood and implemented across different software engineering contexts. He is currently working at Tata Consultancy Services (TCS).



Prof. Dr. Timo Kehrer Timo Kehrer is a Permanent Professor at the University of Bern (Switzerland), where he leads the Software Engineering Research Group (SEG). Kehrer and his group work on a wide range of research topics, including feature-oriented development and software evolution. Kehrer serves on various scientific boards and steering committees, including the Board of Directors of Informatics Europe. He currently serves as President of the Special Interest Group on Software Engineering (CHOOSE) of the Swiss Informatics Society, and is a regular program committee member of leading international software engineering conferences such as ICSE, FSE, ASE, ICSME, SANER, and EASE.



Prof. Dr. Norbert Seyff Norbert Seyff is Professor of Requirements Engineering at the University of Applied Sciences and Arts Northwestern Switzerland (FHNW). His research focuses on Requirements Engineering, sustainability design, end-user involvement, and the role of software engineering in shaping sustainable digital futures, with a strong emphasis on knowledge transfer to industry. He teaches courses on software and requirements engineering as well as sustainable design of digital systems. He is a Council Member of the International Requirements Engineering Board (IREB) and regularly serves as Program Committee Chair and organiser for international conferences such as RE, REFSQ, and ICT4S.

Authors and Affiliations

Nitish Patkar[1] · Aimen Fahmi[2] · Timo Kehrer[3] · Norbert Seyff[1]

✉ Nitish Patkar
nitish.patkar@fhnw.ch

Aimen Fahmi
aimen.fahmi.work@outlook.com

Timo Kehrer
timo.kehrer@unibe.ch

Norbert Seyff
norbert.seyff@fhnw.ch

[1] University of Applied Sciences and Arts Northwestern Switzerland, Windisch, Switzerland

[2] Tata Consultancy Services (TCS), Zürich, Switzerland

[3] University of Bern, Bern, Switzerland