

A Comparative Review and Data-Driven Analysis of Open-Source Software Development Trends

Priyankakumari MiteshKumar Patel*

Abstract

Open-source software (OSS) underpins a substantial portion of modern software infrastructure. Despite its importance, comparative empirical analyses that jointly examine development velocity, popularity–maintenance relationships, and cross-domain differences remain limited. This paper presents a data-driven comparative study of 28 high-impact open-source repositories spanning web frameworks, machine learning systems, infrastructure tooling, and programming language runtimes. Drawing on repository metadata, 7,082 issues, and 248,320 pull requests, we quantify issue resolution times, pull-request merge latency, and efficiency metrics. Our principal findings are fourfold. First, the median pull-request merge time (0.57 days) is substantially lower than the median issue resolution time (15.09 days)—approximately a twenty-six-fold difference. Second, project category exerts a statistically significant effect on both metrics (Kruskal–Wallis $H = 130.81$, $p < 10^{-26}$ for issues; $H = 6547.39$, $p \approx 0$ for pull requests (PRs)). Third, the correlation between stars and open issues is weak (Spearman $\rho = 0.172$, $p = 0.38$), indicating that popularity does not automatically translate into higher maintenance burden. Fourth, efficiency—measured as stars per open issue—varies by more than two orders of magnitude, with Flask emerging as an outlier of exceptional maintenance hygiene (6,747 stars per open issue). These results provide empirical benchmarks for maintainers and contributors and highlight the role of domain-specific governance cultures in shaping development dynamics.

Keywords: Open-source software, empirical software engineering, pull requests, issue resolution, development velocity, comparative analysis

INTRODUCTION

Open-source software (OSS) has transitioned from a niche collaborative practice to the dominant mode of constructing critical software systems. Projects such as the Linux kernel, React, TensorFlow, and Kubernetes form foundational layers of contemporary computing. Foundational empirical work established early quantitative baselines for participation, productivity, and defect density in large open-source projects such as Apache and Mozilla [1]. Understanding how these projects manage contributions, resolve issues, and sustain community health is therefore of both practical and scientific interest.

*Author For Correspondence

Priyankakumari MiteshKumar Patel
E-mail: pmpatel@amrolicollege.ac.in

Ph.D. Scholar, Department of Computer Science, Sabarmati University, Ahmedabad, Gujarat, India

Received Date: September 16, 2026
Accepted Date: September 26, 2026
Published Date: September 28, 2026

Citation: Priyankakumari MiteshKumar Patel. A Comparative Review and Data-Driven Analysis of Open-Source Software Development Trends. Journal of Open Source Developments. 2026; 13(2): 1–9p.

A substantial body of research has examined individual facets of OSS development, such as pull-request acceptance [2, 3], latency [4, 5], social signals in evaluation [3, 6], and issue lifetime [7, 8]. However, comparatively few studies offer a multi-dimensional, cross-category comparison that simultaneously analyzes issue resolution, pull-request throughput, popularity metrics, and community scale within a single coherent framework.

This study addresses this gap. We analyzed 28 carefully selected repositories that represent major domains of modern software development. Our research questions are:

- *RQ1*. How do issue resolution times and pull-request merge times vary across project categories and sizes?
- *RQ2*. What is the relationship between project popularity (stars, forks) and maintenance burden (open issues, contributor load)?
- *RQ3*. Which projects exhibit high efficiency in handling contributions relative to their popularity?
- *RQ4*. What distinctive development patterns emerge when comparing web frameworks, machine learning systems, infrastructure tools, and language runtimes?

The remainder of this paper is organized as follows. The next section reviews related work, followed by a description of the dataset and methodology. The subsequent section presents the results, while the following section discusses the implications and threats to validity. The paper concludes with a summary of the findings.

Related Work

The pull-based development model has been extensively characterized. Gousios et al. [2] provided an early large-scale exploration of pull-request usage on GitHub, reporting high merge rates and rapid turnaround for the majority of contributions. Rahman and Roy [9] complemented this large-scale view with a comparative study of successful and unsuccessful pull requests, identifying discussion text and contributor experience as differentiators of outcome. Subsequent work examined the integrator's and contributor's perspectives [10, 11], revealing tensions around prioritization, quality assurance, and responsiveness. At the level of a single organization, Kononenko et al. [12] examined pull-request merge practices at Shopify's Active Merchant project, illustrating how local process choices shape review outcomes.

Social and technical factors jointly influence contribution outcomes. Tsay et al. [3, 6] showed that evaluators attend both to technical signals (e.g., test presence, commit style) and social signals (e.g., prior interaction, network strength). Dabbish et al. [13] similarly highlighted how the transparency and visibility of collaborator activity on GitHub shapes coordination. Demographic composition further conditions these dynamics: Vasilescu et al. [14] found gender and tenure diversity to be positively associated with team productivity, whereas Terrell et al. [15] documented systematic differences in pull-request acceptance rates between women and men. Latency has received particular attention: Zhang et al. [4, 5] synthesized factors affecting pull-request latency across multiple contexts, Kudrjavets et al. [16] linked smaller code changes to faster merge times across multiple languages, and Soares et al. [17] modelled the factors influencing overall pull-request lifetime, while Hasan et al. [18] focused on time-to-first-response and the growing role of bots.

Issue resolution dynamics have been studied through lifetime modelling [8] and analyses of labelling and assignment practices [7]. Parallel streams of research have investigated continuous integration, code review, and newcomer barriers [19–23].

Despite this rich literature, most studies either examine a single project, a broad unstratified GitHub sample, or a narrow slice of the contribution lifecycle. Our work contributes a focused, multi-category comparative analysis that jointly considers issue and pull-request metrics alongside popularity and community indicators.

METHODOLOGY AND DATASET

Data Collection

Following established practice for pull-based development datasets [24], we assembled a dataset comprising three components for 28 open-source repositories and adhered to recommended precautions for GitHub-derived data throughout collection and cleaning [25].

Repositories were selected to cover major domains: web frameworks (React, Django, Flask, Streamlit), machine learning (TensorFlow, Keras, NumPy, MiniCPM-V), infrastructure (Kubernetes, Terraform, Ansible, Airflow, Elasticsearch, Grafana, Kafka, Dagster, HTTPd), runtimes/languages (Node.js, Swift, Linux, Git), and tooling (Playwright, Webpack, Babel, npm CLI), together with ERPNext, Guava, and Hackingtool. Size categories were assigned as large (5 projects), medium (21 projects), and small (2 projects) based on the overall scale and community size (Table 1).

Metrics and Analysis

Issue resolution time is defined as the interval (in days) between issue creation and closure; open issues are excluded from median calculations. Pull-request merge time is the interval between creation and successful merge. Efficiency is operationalized as stars per open issue. Community indicators include total contributors and open versus closed ratios. We report medians as primary measures of central tendency because both distributions are heavily right-skewed.

Category differences were assessed using the Kruskal–Wallis test. Associations between popularity and maintenance indicators were examined using Spearman’s rank correlation. All analyses were performed using Python (pandas, NumPy, SciPy). Closed issues were used for issue resolution statistics, and merged pull requests were used for merge time statistics. This design avoids treating currently open records as completed events.

RESULTS

Overall Development Velocity

Across the corpus, pull requests are processed markedly faster than issues. The median merge time is 0.57 days, whereas the median issue resolution time is 15.09 days, approximately a twenty-six-fold difference. Mean values are substantially higher (7.16 days for merges; 233.51 days for issues), confirming the presence of long-tailed distributions. The large gap between the median and mean values indicates strongly right-skewed lifetime distributions, with a minority of issues and pull requests remaining open or under review for extended periods (Table 2).

Variation by Category

Project category has a statistically significant effect on both issue resolution time (Kruskal–Wallis $H = 130.81$, $p < 10^{-26}$) and pull-request merge time ($H = 6547.39$, $p \approx 0$) (Table 3 and Figure 1).

Table 1. Dataset summary.

Component	N	Primary fields
Repository metadata	28	Stars, forks, open issues, contributors, size category
Issues	7,082	State, creation/closure timestamps, resolution time
Pull requests	248,320	State, creation/merge timestamps, merge time

Table 2. Aggregate velocity statistics.

Metric	Value
Total issues	7,082
Closed issues	3,872 (54.7%)
Median issue resolution time	15.09 days
Mean issue resolution time	233.51 days
Total pull requests	248,320
Merged PRs	183,836 (74.0%)
Median PR merge TIME	0.57 days
Mean PR merge Time	7.16 days

Testing and tooling projects exhibited the fastest issue resolution (median 5.26 days), followed by enterprise software (8.71 days). Web frameworks and libraries showed substantially longer medians. For pull requests, libraries and tooling again led, while runtime and language projects (notably Linux and Git) displayed longer merge times consistent with rigorous review cultures required for core system software (Figure 2 and Table 4).

Popularity and Maintenance Burden

A common assumption is that higher popularity entails a heavier maintenance load. Our data provide only weak support for this claim. The Spearman correlation between stars and open issues is $\rho = 0.172$ ($p = 0.38$), indicating a weak and non-significant relationship. In contrast, stars and forks are strongly associated ($\rho = 0.817$, $p < 0.001$) (Figure 3).

Table 3. Median issue resolution time by category.

Category	Median (days)
Testing/tooling	5.26
Enterprise	8.71
Runtime/language	19.08
Infrastructure	25.57
Machine learning	26.58
Web framework	67.27
Library	252.11

Table 4. Median PR merge time by category.

Category	Median merge time (days)
Library	0.017
Testing/Tooling	0.124
Enterprise	0.234
Infrastructure	0.621
Web Framework	0.636
Machine Learning	0.917
Runtime/Language	2.003

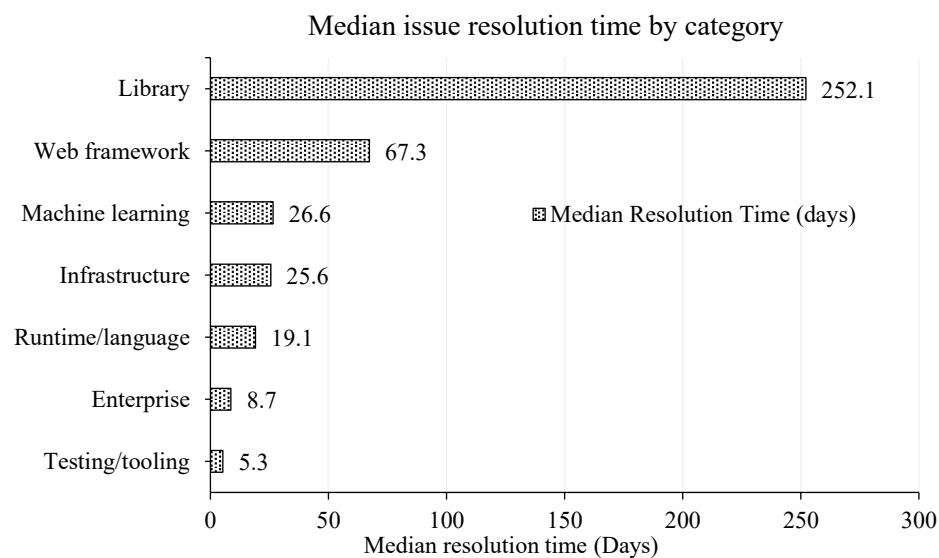


Figure 1. Median issue resolution time (days) by project category.

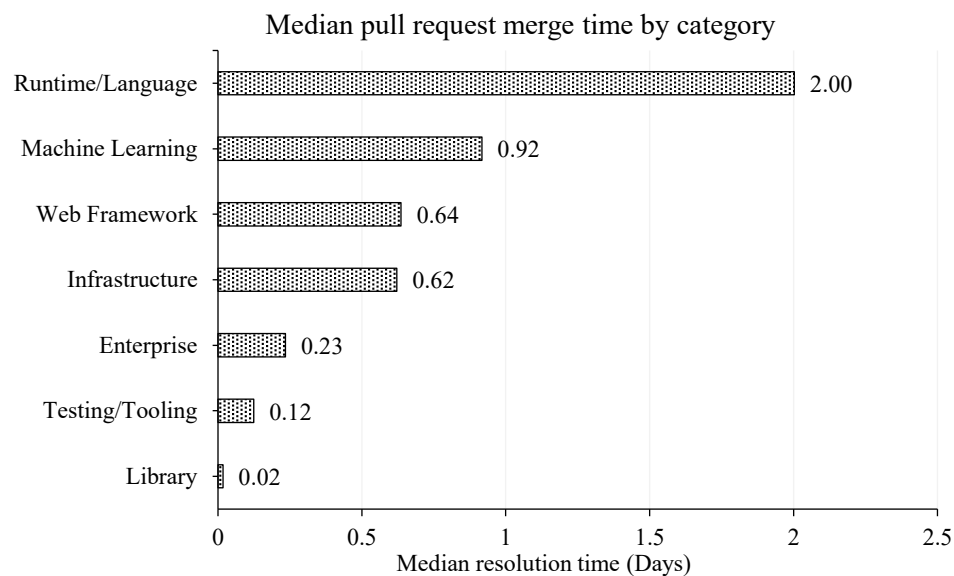


Figure 2. Median pull-request merge time (days) by project category.

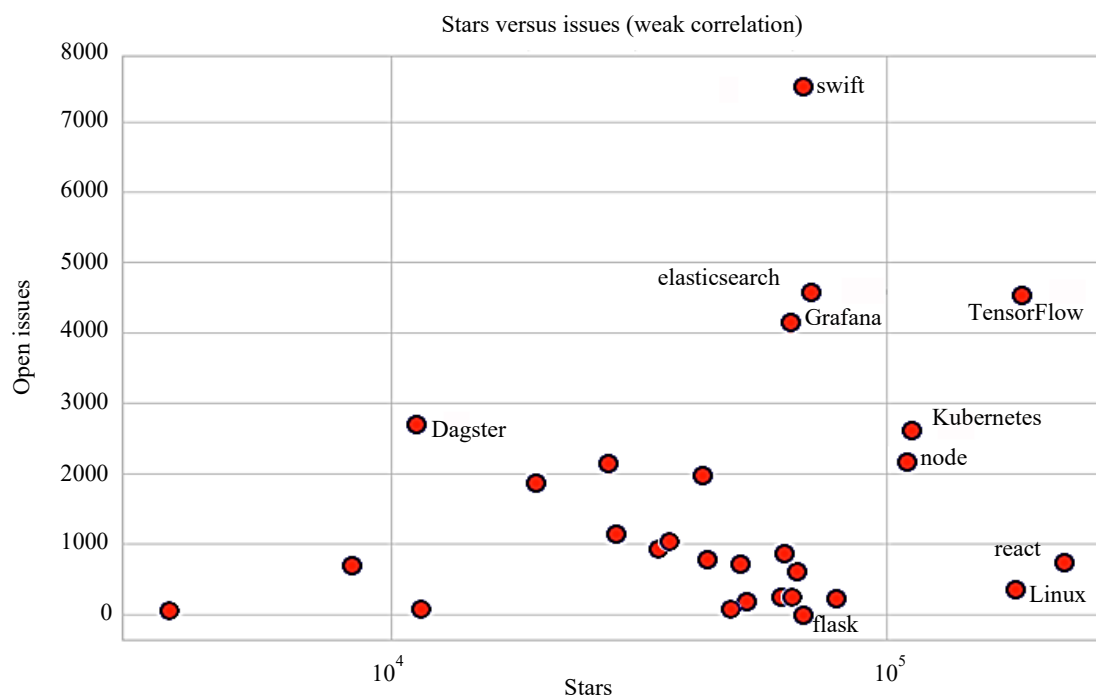


Figure 3. Relationship between stars and open issues (logarithmic x-axis).

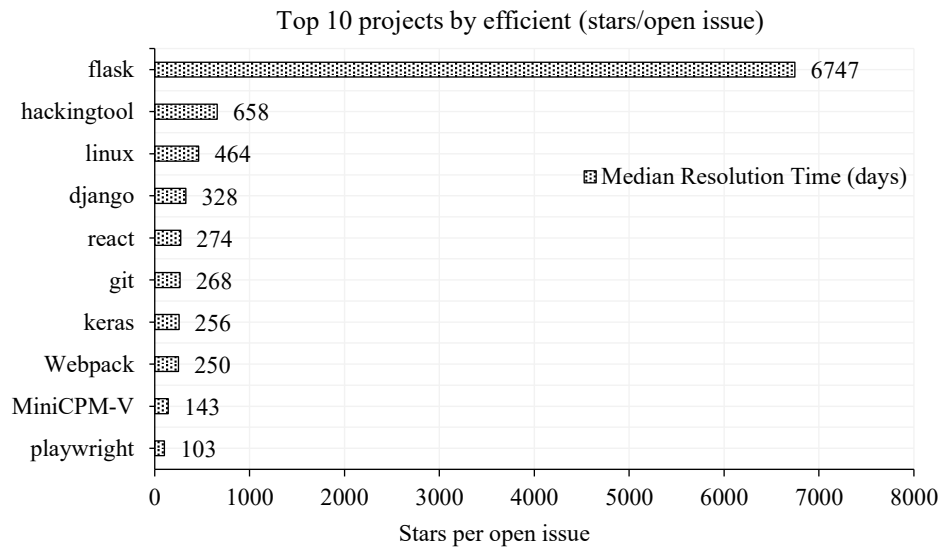
This finding challenges the common assumption that popular projects inevitably accumulate large backlogs of unresolved issues. Project governance, community norms, and tooling appear to play a larger role than raw popularity.

Efficiency Rankings

We define efficiency as the ratio of stars to open issues. Higher values indicate that a project maintains a relatively clean issue tracker relative to its popularity. Flask achieves an exceptional score of approximately 6,747 stars per open issue, far exceeding other projects. Linux, Django, and React also rank highly, while several infrastructure and machine learning projects exhibit lower ratios (Table 5).

Table 5. Top projects by stars per open issue efficiency.

Project	Stars	Open issues	Stars/issue
Flask	67,474	10	6,747
Hacking tool	48,056	73	658
Linux	177,813	383	464
Django	79,035	241	328
React	226,971	829	274

**Figure 4.** Top projects ranked by stars per open issue.

Flask stands out dramatically. Despite nearly 68,000 stars, it maintains only ten open issues at the time of data collection, yielding an efficiency score more than an order of magnitude higher than most other projects (Figure 4).

Project Level Extremes

At the individual project level, the fastest median issue resolution times were observed for Flask (0.06 days), Babel (0.67 days), and MiniCPM-V (3.33 days). The slowest were Guava (252 days), TensorFlow (231 days), and Grafana (110 days).

For pull requests, Guava, Playwright, and Elasticsearch showed median merge times under 0.15 days, while Linux (261 days) and Git (24 days) exhibited the longest median merge times—reflecting the deliberate, high-assurance review cultures of these foundational projects.

DISCUSSION

Principal Observations

Several important trends emerge from the analysis:

- *Speed asymmetry.* Pull requests are merged far more quickly than issues are resolved. This suggests that code contributions receive higher priority or benefit from clearer acceptance criteria compared with general issue discussions.
- *Category matters.* Domain and project types strongly influence development speed. Tooling and library projects often exhibit faster cycles than complex infrastructure or language projects.
- *Popularity $\neq$ burden.* The weak correlation between stars and open issues challenges the assumption that popular projects necessarily suffer from an overwhelming maintenance load. Effective governance and community processes appear to be more decisive.
- *Governance signal.* Extremely long median merge times in projects such as Linux and Git reflect deliberate, high-assurance review processes rather than inefficiency.

Implications

These findings have practical implications. New contributors may prefer projects with faster feedback cycles (e.g., many tooling projects). Maintainers can benchmark their own resolution and merge times against category peers. Researchers gain a clearer picture of the diversity of development cultures across the open-source landscape. For contributors, projects in the tooling and library categories currently offer the shortest feedback cycles. For maintainers, the efficiency metric and category benchmarks supply reference points against which internal processes can be compared. For researchers, the results underscore the value of stratified, multi-metric designs over single project or unstratified analyses.

Threats to Validity

Several limitations should be noted. First, the analysis covers 28 repositories and therefore should not be interpreted as representative of the entire OSS ecosystem; the sample, while diverse, is not a random sample of GitHub and over-represents high-visibility projects. Second, repository categories are broad and may contain substantial within-category variation. Third, the analysis relies on issue and pull-request metadata and does not directly model code review discussion quality, issue-topic semantics, contributor retention, or governance practices. Issue and pull-request data reflect only the GitHub interface; off-platform discussion and mergers are not captured. Resolution and merge times are sensitive to project-specific conventions (e.g., prolonged “on hold” states). Fourth, repository popularity indicators such as stars and forks are observational measures and should not be interpreted as causal drivers of maintenance behavior. Finally, the cross-sectional nature of the snapshot precludes strong causal claims about longitudinal trends; repository activity changes over time, and the reported values represent a snapshot rather than permanent project characteristics.

Data Availability

The dataset used in this study, including repository metadata, issue records, and pull-request records for the 28 selected open-source projects, along with the analysis scripts (Python) used to compute resolution times, merge times, and statistical tests, is publicly available at: <https://github.com/pmpatel-cpu/Open-Source-Software-Development-Trends>. All underlying data were collected from publicly accessible GitHub repositories in accordance with GitHub’s terms of service and established mining guidelines [25].

CONCLUSION AND FUTURE WORK

This study presents a comparative, data-driven analysis of development trends across 28 prominent open-source repositories. By jointly examining issue resolution, pull-request latency, popularity, and efficiency, we have shown that development velocity varies systematically by category, that popularity is only weakly coupled to open-issue burden, and that certain projects achieve markedly higher maintenance efficiency than their peers. The results indicate that project category is strongly associated with issue resolution and pull-request merge times, while repository popularity alone does not provide a sufficient explanation for maintenance burden.

Future work should incorporate longitudinal data across multiple years to examine how development and maintenance patterns evolve overtime. Additional community-level variables, including code review comment volume, contributor retention, and contributor activity, can be incorporated to provide a more comprehensive understanding of project health. The study can also be extended to a larger and more balanced set of repositories across additional software categories.

Finally, qualitative and text-based analysis of issue and pull-request titles, descriptions, and discussions could be used to identify recurring themes and development trends that are not captured by quantitative repository metrics alone.

REFERENCES

1. Mockus A, Fielding RT, Herbsleb JD. Two case studies of open source software development: Apache and Mozilla. *ACM Trans Softw Eng Methodol.* 2002;11(3):309–346. doi:10.1145/567793.567795.
 2. Gousios G, Pinzger M, van Deursen A. An exploratory study of the pull-based software development model. 36th International Conference on Software Engineering (ICSE 2014), Hyderabad, India. 2014. p.345–55. doi:10.1145/2568225.2568260.
 3. Tsay J, Dabbish L, Herbsleb J. Influence of social and technical factors for evaluating contribution in GitHub. 36th International Conference on Software Engineering (ICSE 2014), Hyderabad, India. 2014. p.356–66. doi:10.1145/2568225.2568315.
 4. Zhang X, Yu Y, Wang T, Rastogi A, Wang H. Pull request latency explained: an empirical overview. *Empir Softw Eng.* 2022;27(6):126. doi:10.1007/s10664-022-10143-4.
 5. Zhang X, Yu Y, Gousios G, Rastogi A. Pull request decisions explained: an empirical overview. *IEEE Trans Softw Eng.* 2023;49(2):849–871. doi:10.1109/TSE.2022.3165056.
 6. Saroar SG, Nayebi M. Developers' perception of GitHub Actions: a survey analysis. 27th International Conference on Evaluation and Assessment in Software Engineering (EASE 2023), Oulu, Finland. 2023. p.121–30. doi:10.1145/3593434.3593475.
 7. Alam K, Roy B. What drives issue resolution speed? An empirical study of scientific workflow systems on GitHub. 2026 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), Auckland, New Zealand. 2026. p.1–6. doi:10.1109/SANER67736.2026.00116.
 8. Kikas R, Dumas M, Pfahl D. Using dynamic and contextual features to predict issue lifetime in GitHub projects. 13th International Conference on Mining Software Repositories (MSR 2016), Austin, TX, USA. 2016. p.291–302. doi:10.1145/2901739.2901751.
 9. Rahman MM, Roy CK. An insight into the pull requests of GitHub. 11th Working Conference on Mining Software Repositories (MSR 2014), Hyderabad, India. 2014. p.364–7. doi:10.1145/2597073.2597121.
 10. Gousios G, Zaidman A, Storey MA, van Deursen A. Work practices and challenges in pull-based development: the integrator's perspective. 2015 IEEE/ACM 37th International Conference on Software Engineering (ICSE), Florence, Italy. 2015. p.358–68. doi:10.1109/ICSE.2015.55.
 11. Li Z, Yu Y, Wang T, Yin G, Li S, Wang H. Are you still working on this? An empirical study on pull request abandonment. *IEEE Trans Softw Eng.* 2022;48:2173–2188. doi:10.1109/TSE.2021.3053403.
 12. Kononenko O, Rose T, Baysal O, Godfrey MW, Theisen D, de Water B. Studying pull request merges: a case study of Shopify's active merchant. 2018 IEEE/ACM 40th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), Gothenburg, Sweden. 2018. p.124–33. doi:10.1145/3183519.3183542.
 13. Dabbish L, Stuart C, Tsay J, Herbsleb J. Social coding in GitHub: transparency and collaboration in an open software repository. *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*, Seattle, WA, USA. 2012. p.1277–86. doi:10.1145/2145204.2145396.
 14. Vasilescu BN, Posnett D, Ray B, van den Brand MGJ, Serebrenik A, Devanbu P, et al. Gender and tenure diversity in GitHub teams. 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI 2015), Seoul, Republic of Korea. 2015. p.3789–98. doi:10.1145/2702123.2702549.
 15. Terrell J, Kofink A, Middleton J, Raine C, Murphy-Hill E, Parnin C, et al. Gender differences and bias in open source: pull request acceptance of women versus men. *PeerJ Comput Sci.* 2017;3:e111. doi:10.7717/peerj-cs.111.
 16. Kudrjavets G, Nagappan N, Rastogi A. Do small code changes merge faster? A multi-language empirical investigation. 2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR), Pittsburgh, PA, USA. 2022. p.537–48. doi:10.1145/3524842.3528448.
 17. Moreira Soares D, de Lima Júnior ML, Murta L, Plastino A. What factors influence the lifetime of pull requests? *Softw Pract Exp.* 2021;51(6):1173–1193. doi:10.1002/spe.2946.
-

18. Hasan KA, Macedo M, Tian Y, Adams B, Ding S. Understanding the time to first response in GitHub pull requests. 2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR), Melbourne, Australia. 2023. p.1–11. doi:10.1109/MSR59073.2023.00015.
19. Vasilescu B, Yu Y, Wang H, Devanbu PT, Filkov V. Quality and productivity outcomes relating to continuous integration in GitHub. Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2015), Bergamo, Italy. 2015. p.805–16. doi:10.1145/2786805.2786850.
20. Yu Y, Wang H, Filkov V, Devanbu P, Vasilescu B. Wait for it: determinants of pull request evaluation latency on GitHub. 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories (MSR), Florence, Italy. 2015. p.367–71. doi:10.1109/MSR.2015.42.
21. Zampetti F, Scalabrino S, Oliveto R, Canfora G, Di Penta M. How open source projects use static code analysis tools in continuous integration pipelines. 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR), Buenos Aires, Argentina. 2017. p.334–44. doi:10.1109/MSR.2017.2.
22. Baysal O, Kononenko O, Holmes R, Godfrey MW. The influence of non-technical factors on code review. 2013 20th Working Conference on Reverse Engineering (WCRE), Koblenz, Germany. 2013. p.122–31. doi:10.1109/WCRE.2013.6671287.
23. Steinmacher I, Conte TU, Gerosa MA, Redmiles DF. Social barriers faced by newcomers placing their first contribution in open source software projects. 18th ACM Conference on Computer Supported Cooperative Work & Social Computing (CSCW), Vancouver, Canada. 2015. p.1379–92. doi:10.1145/2675133.2675215.
24. Gousios G, Zaidman A. A dataset for pull-based development research. 11th Working Conference on Mining Software Repositories (MSR), Hyderabad, India. 2014. p.368–71. doi:10.1145/2597073.2597122.
25. Kalliamvakou E, Gousios G, Blincoe K, Singer L, German DM, Damian D. The promises and perils of mining GitHub. 11th Working Conference on Mining Software Repositories (MSR), Hyderabad, India. 2014. p.92–101. doi:10.1145/2597073.2597074.