Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Removing Noise or Introducing Bias? The Hidden Cost of MSR Filtering

Mohit Kaushik

mohitcs.rsh@gndu.ac.in

Guru Nanak Dev University https://orcid.org/0009-0008-1294-7337

Jyoti Bawa

Guru Nanak Dev University

Research Article

Keywords: MSR, Sampling Biases, Open Source Software

Posted Date: July 28th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-10495510/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: The authors declare no competing interests.

Removing Noise or Introducing Bias? The Hidden Cost of MSR Filtering

Mohit Kaushik
Department of Computer Science
Guru Nanak Dev University
Amritsar, India
mohitcs.rsh@gndu.ac.in

Jyoti Bawa
Department of Computer Science
Guru Nanak Dev University
Amritsar, India
jyoticsc.rsh@gndu.ac.in

***Abstract*—OSS platforms like GitHub serve as a primary data source for MSR research. As the platform is widely used by different users, spanning from student to developer, not all repositories are actual engineered projects. Therefore, to avoid such noise, researchers often apply several criteria, which may fundamentally change the sample demographics. To understand such biases, this study aims to uncover the hidden cost originating from these arbitrary thresholds or criteria. We analyzed 1.57 million repositories from the SEART platform and constructed several datasets from the thresholds often applied in MSR research. We identify the maintenance bias in these filtering processes, which masks the true abandonment (73.42%) realities of OSS projects. Also, these strategies favor some ecosystems and governance styles. Moreover, the sampling strategy also distorts the relationship between variables, suffering from relational biases. Therefore, to avoid such biases, the researchers should shift towards stratified sampling and refining the criteria for noise detection.**

***Index Terms*—MSR, Sampling Biases, Open Source Software**

I. Introduction

Open Source Software (OSS) platforms like GitHub are fueling the software engineering (SE) research. From an analysis of ≈10,000 SE papers, 20-25% of papers actually utilize GitHub artifacts such as commits, PRs, Issues, stars, etc [1]. This proportion is even larger for top mining software repository (MSR) venues [1]. Approximately 40-50% of studies from venues like ICSE, ASE, and FSE use GitHub data [1]. As of 2026, GitHub officially hosts 630 million repositories [2], which make it the primary data source for SE or MSR studies. But not all repositories are a valid data source for research.

GitHub is widely used by students, teachers, or instructors for experimentation purposes [3], [4]. As a result, the researchers usually exclude these experimental or toy repositories from their analysis [5]. Considering such repositories as noise, the researchers usually apply certain noise-removing filters [3]. These filters include, but are not limited to, popularity-based or activity-based filters [4]. For example, some studies use a generic threshold of a minimum of 10 stars (a proxy for project popularity on GitHub) to remove noise [5]. But this 10-star baseline still leaves massive noise. To further handle this noise, the researchers apply a secondary, higher star threshold, for example, selecting the repositories having more than 500 stars [4]. Although it actually helps to combat noise, it completely changes the sample demographics, for instance, favoring a certain governance style or ecosystem [6]. Moreover, it can change the statistical relationship, thereby limiting the results' generalizability to their own platform [7].

Therefore, this study aims to analyze the change in sample demographics by these arbitrary filters. In this study we answer how these noise-reduction thresholds do not just clean the data but systematically distort the reality of the OSS ecosystem. Specifically, we aim to analyze the composition, maintenance, and relational biases that originated from these arbitrary thresholds.

*Presented at the International Conference on Computing Innovations for Sustainability (ICCIS), Guru Nanak Dev University, Amritsar, India.

II. Related Works

Historically, SourceForge was the primary data source for MSR research. Foundational studies such as those by Howison and Crowston [8] highlighted some metadata inconsistencies in SourceForge projects. The SourceForge projects suffered from survivorship bias, where only 1% of projects were actually active [9]. Later on, GitHub was introduced in 2008. The prime GitHub data sources, such as GHTorrent [10] and GHArchive [11], were introduced and launched in 2012. These platforms host thousands to millions of repositories' data, thereby providing ease to the MSR researchers. However, some researchers were blindly treating all GitHub repositories as legitimate software engineering projects.

Kalliamvakou et al.'s study [3], presented in MSR 2014, noted 13 specific perils (dangers) to blindly mining or using GitHub for research. They specifically highlighted that 37% of analyzed projects were actually noise or experimental and a vast majority of projects were dead. Following their established perils, the MSR researchers are now applying specific noise filters, such as setting a threshold on project stars, commits, pull requests, etc [4], [5]. However, these filters introduce severe biases. Tutko et al. [12], in their systematic literature review of 286 recent MSR studies, highlighted that ≈86.7% of MSR papers rely on non-probability sampling. 43.7% of the sampled papers utilized the popularity-based criteria to avoid noise or trivial repositories.

Despite the high popularity of star-based filters, they suffer from severe data loss. To demonstrate this, Munaiah et al. [4] conducted a study on identifying real software engineer projects. They highlighted that using a 10-star threshold exhibited a high precision of 97% with a terrible recall of only 32%. The authors further tested their model on 50 and 500 star thresholds and noted the 14% and 0% recall. Therefore, relying on a star-based threshold to identify a real software engineering project is risky, as the same authors highlighted a loss of 68% of genuine software projects. Even our pilot study using GitHub's advanced search found 317 million repositories that have fewer than 10 stars. Our manual check found that some of them were highly active with more than 500 commits. Additionally, the number of the stars on a project does not equate to their actual usage. Koch et al. [13] revealed a weak correlation between number of stars and actual downloads. As a result, the star-based thresholds are lossy and misleading in nature.

Considering the potential biases with star-based thresholds, some studies adopted activity-based criteria. For instance, Xiao et al. [14] applied a 95th-percentile activity threshold by excluding the repositories having fewer than 57 commits. Similarly, Ray et al. [15] also retain the repositories satisfying the threshold of 20 commits to isolate the active projects from noise. Raw commit volume suffers from accumulation bias and is further inflated by bots [16]. Kaushik and Chahal's study [17] specifically identified that monthly rates of commits are highest in younger, short-lived projects and relying on higher commit-activity thresholds, the researchers are sampling short-lived projects that have an initial burst of commit activity. Therefore, the MSR studies should not rely on the assumption that high commit activity is always associated with longer-lived and healthier projects [17]. Moreover, these commit-based thresholds not only clean the dataset by removing experimental noise but also change the statistical relationship between variables. Thakur and Mockus [6] revealed a flip in the relationship between project activity and popularity after applying stricter commit-based thresholds. As these thresholds inadvertently change the relationship between variables, models derived from these thresholds may introduce bias or limit generalizability.

III. Methodology

This section presents our research questions, data source, and methodology.

A. Research Questions

Our study is guided by three research questions, outlined below.

- RQ1: How do popularity and activity-based thresholds alter the representation of programming language, repository age, and project size?
- RQ2: How do the arbitrary commit and star thresholds change the active project age and project abandonment rate?
- RQ3: To what extent do these noise filters change the statistical relationship between developer activity and project popularity?

B. Data Collection, Preprocessing and Construction

To initiate the analysis, we collected the dataset from the SEART platform[1]. This platform was introduced by Dabic et al. [5] in MSR 2021. To handle noise and support valid repository sampling, the platform natively applied a 10-star threshold. We downloaded a dataset of 1.9 million repositories across 35 attributes. Following this, we removed the null values for commits, total issues, and contributors. Moreover, we removed the forked repositories to avoid duplication and excluded empty projects (size = 0 KB). This process yielded a clean baseline dataset of 1.57 million repositories.

For dataset construction, we applied both popularity- and activity-based thresholds. As illustrated in Figure 1, we followed a two-pronged approach. For RQ1 and RQ2, we applied various thresholds on the baseline dataset, outlined below.

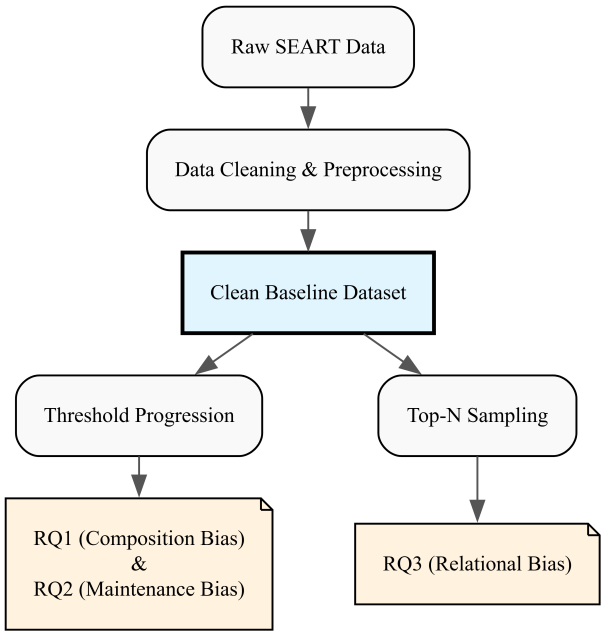


Fig. 1. An Overview of the research approach followed in this work

- Popularity-based thresholds: We tested the sample composition and maintenance bias by varying the star-based thresholds from 10 to 1000 (10, 20, 50, 100, 500, 1000). With these thresholds, we computed the projects' programming language compositions, license proportions, median size, median team size, and median age.
- Activity-based thresholds: Similarly, we varied the commit threshold from 10 to 1000 (10, 20, 50, 100, 500, 1000).

For RQ3, we constructed the three datasets: Dataset A, B, and C. Dataset A contains the top 10,000 most popular

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

projects, whereas Dataset B contains the top 10,000 most active projects (top commits). Dataset C also contains 10,000 randomly sampled repositories stratified by programming language . All these datasets were constructed from a baseline dataset of 1.57 million repositories. To answer RQ3, we compared these datasets' attributes' correlations to reveal how these criteria inflate or degrade the correlation between the underlying attributes.

IV. Results

This section presents the results of this study. We organized these results in accordance with the proposed research questions. Table I summarizes the results for RQ1 and RQ2.

Popularity-based thresholds (Table I, upper portion) highlight the governance and maintenance biases. As the popularity-based threshold increases, the project size increases substantially by 7x. This indicates that a sample of higher-popularity projects favors large projects. In a similar manner, median contributors' size and project age also scale gradually. Median creation year also revealed a shift from 2020 to 2018. Regarding maintenance bias, the baseline noted 73.42% inactive projects favoring the active projects by continuously reducing the inactive projects' proportions. Moreover, the higher popularity thresholds favor permissive projects, continuously dropping the informal, ad hoc projects with no license. In the case of ecosystem share, Python (20%) and JavaScript (13-14%) hold steadily as the threshold progresses. TypeScript's proportion increased from 9 to 12%. Go, which is not in the top five programming languages at lower thresholds (10-20 stars), increased its proportion to 7.6% at 1000 stars. Conversely, the C++ present in lower thresholds ($\leq$ 50) drops out of the top 5 after $\geq$ 500 stars. Overall, this analysis confirms that the higher popularity thresholds favor older, large, permissive, and infrastructural projects, which attract higher popularity and longevity.

Similarly, for median project size, team size, and project age, the commit-based threshold (Table I, lower portion) revealed a similar behavior. As the commit threshold increases, the project size is increased by $\approx$18x. Moreover, the median number of contributors and project age also steadily increased from 3 to 19 and 29.50 months to 68.63 months, respectively. Regarding governance, the commit-based threshold favors the copyleft license by dropping permissive license shares from 51.49% to 47.63%. Copyleft's share is increased from 11.47% to 20.22% on higher thresholds. Moreover, inactive projects' proportions continuously dipped from 72.12% to 40.49%. These activity-based thresholds also favor certain ecosystems. Python (20%-17%) and JavaScript (14%-10.8%) proportions gradually decline at higher thresholds. TypeScript (9.4%-12.7%) and Java (5.4%-8.4%) proportions increased steadily with higher commit thresholds. C++ nearly doubled its share from 5.36% to 9.51%. Similar to star-based thresholds, the commit-based thresholds also favor older, large infrastructural projects with larger teams but support restrictive copyleft licenses. Table II reports the Spearman correlations for the developer activity and popularity attributes across all three datasets. We applied Spearman because the OSS dataset, specifically from GitHub, inherently exhibits a non-linear relationship, and the distribution-agnostic nature of Spearman makes it suitable for this dataset.

In the baseline (Dataset C), all the correlations are weak to moderate. However, filtering significantly warps these relationships. The correlation of commits with size is inflated in both popular and active projects. This is expected, as both thresholds favor large projects. Moreover, the relationships between commits and total issues as well as total issues and contributors are inflated in the popularity-filtered sample (0.707 and 0.692) but degraded in the activity-filtered sample (0.331 and 0.365). Also, the correlations with stargazers degrade independently of the filtering process. Overall, these findings confirmed the presence of relational bias. Applying top-N filtering fundamentally distorts the relationship between software metrics.

V. Discussion and Limitations

Our study aims to highlight the potential trade-offs between noise and bias. MSR studies usually adopt arbitrary thresholds to eliminate noise. These criteria, while successfully removing the noises, introduce several biases, which in turn may lead to a generalizability crisis [7]. Our results highlight how adopting a filtering strategy favors certain ecosystems and governance, which masks the OSS ecosystem's messy realities. The statistical relationship varied substantially across these filtering techniques. If a study with top-N filtering trains a predictive model, the model will likely learn the sampling artifacts rather than actual software engineering principles and relationships. These findings are aligned with Kalliamvakou et al. [3], who first highlight the OSS projects' reality by highlighting promise and perils. Similar concerns were also raised by Tutko et al. [12] that removing noise fundamentally changes the relationship between variables.

Limitations:: Our study has certain limitations. First, we relied on the SEART platform for a data source, which may contain data biases. The dataset, which we considered as a baseline, natively satisfies the 10-star threshold, which completely avoids the absolute bottom tail of GitHub projects. Moreover, as our pilot study noted, unpopular repositories are not always noise. Second, we considered the commits and stargazers as proxies for activity and popularity. These metrics natively suffer from accumulation bias and are mediated by bots' presence [16].

VI. Conclusion and Future Work

This study highlighted several severe biases originating from filtering processes. Our results highlighted that different thresholds distort the dataset demographics. Star-based thresholds artificially inflated the project size by 7x, whereas commit-based thresholds increased it by 18x. Additionally, star-based thresholds favored the permissive licenses and infrastructural projects. On the other hand, commit-based thresholds favored different governance styles. Moreover, both threshold criteria inflated or degraded the correlation, thereby

TABLE I
THRESHOLD PROGRESSION ANALYSIS FOR POPULARITY (STARS) AND ACTIVITY (COMMITS)

RQ	Metric	Stars Thresholds					
		≥ 10	≥ 20	≥ 50	≥ 100	≥ 500	≥ 1000
RQ1	Size (KB)	1,360.0	1,719.0	2,420.0	3,221.0	6,648.5	9,477.0
	Contributors	2	3	4	6	14	21
	Age (Months)	26.74	32.29	39.26	44.94	60.25	67.05
RQ2	Permissive (%)	50.12	52.91	56.01	58.15	62.24	63.18
	No License (%)	30.03	26.19	21.85	18.75	12.20	10.18
	Copyleft (%)	11.01	11.26	11.58	11.89	12.59	12.93
	Inactive (%)	73.42	71.13	67.79	64.73	55.52	50.65
	Creation Date	2020-02	2019-09	2019-04	2019-01	2018-05	2018-02

RQ	Metric	Commits Thresholds					
		≥ 10	≥ 20	≥ 50	≥ 100	≥ 500	≥ 1000
RQ1	Size (KB)	1,548.0	1,876.0	2,846.0	4,387.0	15,517.0	28,335.5
	Contributors	3	3	4	5	12	19
	Age (Months)	29.50	32.88	39.19	45.20	61.32	68.63
RQ2	Permissive (%)	51.49	52.65	53.85	53.96	50.61	47.63
	No License (%)	27.92	25.72	22.17	19.63	15.32	13.91
	Copyleft (%)	11.47	12.08	13.39	14.70	18.41	20.22
	Inactive (%)	72.12	69.97	65.03	59.82	45.80	40.49
	Creation Date	2020-01	2019-12	2019-11	2019-10	2019-04	2018-11

TABLE II
SPEARMAN CORRELATIONS ACROSS DATASETS (RELATIONAL BIAS)

Variable Pair	Dataset C Baseline	Dataset A Popularity Filter	Dataset B Activity Filter
commits vs. size	0.466	0.603↑	0.808↑
commits vs. totalIssues	0.517	0.707↑	0.331↓
commits vs. contributors	0.588	0.789↑	0.720↑
totalIssues vs. contributors	0.568	0.692↑	0.365↓
totalIssues vs. stargazers	0.499	0.323↓	0.369↓
totalIssues vs. size	0.205	0.435↑	0.284↑
contributors vs. size	0.177	0.375↑	0.624↑
contributors vs. stargazers	0.359	0.321↓	0.228↓
commits vs. stargazers	0.282	0.234↓	0.243↓
size vs. stargazers	0.159	0.192↑	0.241↑

↑ = inflated correlation, ↓ = degraded correlation.

introducing correlation bias. Due to these biases, the MSR researchers must stop applying these arbitrary, top-N thresholds without sensitivity analysis or proper justification. Instead, future research should adopt stratified random sampling approaches to ensure that removing noise does not destroy the sample representativeness. Moreover, the definition and criteria to identify noise could be refined to avoid dropping real engineered projects.

REFERENCES

[1] K. Alrashedy and A. Binjahlan, "How do software engineering researchers use github? an empirical study of artifacts & impact," in *2024 IEEE International Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 2024, pp. 118–130. [Online]. Available: https://doi.org/10.1109/SCAM63643.2024.00021

[2] G. Staff, "Octoverse 2025: A new developer joins github every second as ai leads typescript to #1," https://gh.io/octoverse2025, October 2025, accessed: May 23, 2026.

[3] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, "An in-depth study of the promises and perils of mining github," *Empirical Software Engineering*, vol. 21, no. 5, pp. 2035–2071, 2016.

[4] N. Munaiah, S. Kroh, C. Cabrey, and M. Nagappan, "Curating github for engineered software projects," *Empirical Software Engineering*, vol. 22, no. 6, pp. 3219–3253, 2017. [Online]. Available: https://link.springer.com/article/10.1007/s10664-017-9512-6

[5] O. Dabic, E. Aghajani, and G. Bavota, "Sampling projects in github for msr studies," in *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. IEEE, 2021, pp. 560–564.

[6] A. Malviya-Thakur and A. Mockus, "The role of data filtering in open source software ranking and selection," in *Proceedings of the 1st IEEE/ACM International Workshop on Methodological Issues with Empirical Studies in Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2024, pp. 7–12. [Online]. Available: https://doi.org/10.1145/3643664.3648210

[7] S. Baltes and P. Ralph, "Sampling in software engineering research: A critical review and guidelines," *Empirical Software Engineering*, vol. 27, no. 4, p. 94, 2022. [Online]. Available: https://link.springer.com/article/10.1007/s10664-021-10072-8

[8] J. Howison and K. Crowston, "The perils and pitfalls of mining sourceforge," in *Proceedings of the 26th International Conference on Software Engineering - W17S Workshop: International Workshop on Mining Software Repositories (MSR 2004)*. IET, 2004, pp. 7–11.

[Online]. Available: https://doi.org/10.1049/ic:20040467
[9] A. Rainer and S. Gale, "Evaluating the quality and quantity of data on open source software projects," in *Proceedings of the 1st International Conference on Open Source Systems (OSS 2005)*, 2005, pp. 29–36. [Online]. Available: https://uhra.herts.ac.uk/id/eprint/13709/
[10] G. Gousios and D. Spinellis, "Ghtorrent: Github's data from a firehose," in *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*. IEEE, 2012, pp. 12–21.
[11] I. Grigorik, "Gh archive: Github public timeline data," https://www.gharchive.org/, 2012, accessed: May 23, 2026.
[12] A. Tutko, A. Z. Henley, and A. Mockus, "How are software repositories mined? a systematic literature review of workflows, methodologies, reproducibility, and tools," *arXiv preprint arXiv:2204.08108*, 2022. [Online]. Available: https://arxiv.org/abs/2204.08108
[13] S. Koch, D. Klein, and M. Johns, "The fault in our stars: An analysis of github stars as an importance metric for web source code," in *Workshop on Measurements, Attacks, and Defenses for the Web (MADWeb)*, 2024, pp. 65–76. [Online]. Available: https://www.ias.cs.tu-bs.de/publications/GithubTranco.pdf
[14] W. Xiao, H. He, W. Xu, Y. Zhang, and M. Zhou, "How early participation determines long-term sustained activity in github projects?" in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2023, pp. 29–41. [Online]. Available: https://doi.org/10.1145/3611643.3616349
[15] B. Ray, D. Posnett, V. Filkov, and P. Devanbu, "A large scale study of programming languages and code quality in github," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2014, pp. 155–165. [Online]. Available: https://doi.org/10.1145/2635868.2635922
[16] T. Dey, B. Vasilescu, and A. Mockus, "An exploratory study of bot commits," in *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*. New York, NY, USA: Association for Computing Machinery, 2020, pp. 61–65. [Online]. Available: https://doi.org/10.1145/3387940.3391502
[17] M. Kaushik and K. K. Chahal, "Community engagement and the lifespan of open-source software projects," *Information and Software Technology*, vol. 189, p. 107914, 2026. [Online]. Available: 10.1016/j.infsof.2025.107914