Understanding the Popularity of Packages in Maven Ecosystem

Sadman Jashim Sakib * , Muhammad Asaduzzaman , Curtis Bright , Cole Morgan

Posted Date: 28 April 2025

doi: 10.20944/preprints202504.2296.v1

Keywords: Popularity Metrics; Software Packages; Maven; GitHub

 doi:10.20944/preprints202504.2296.v1

Article

Understanding the Popularity of Packages in Maven Ecosystem

Sadman Jashim Sakib *, Muhammad Asaduzzaman, Curtis Bright and Cole Morgan

School of Computer Science, University of Windsor, Windsor, ON, Canada
* Correspondence: sakib51@uwindsor.ca

Abstract: The widespread availability of open-source software packages in ecosystems like Maven has significantly improved developer productivity by promoting the reuse of pre-existing packages. However, the vast number of available packages often poses challenges in selecting suitable packages. This study investigates the role of popularity metrics in evaluating Maven packages by analyzing 103,315 packages, each at least two years old. Metrics were collected from the Maven Neo4j dataset and GitHub repositories to examine their relationships and importance in determining package popularity. Our analysis reveals strong interdependencies among community-driven GitHub metrics, such as stars, forks, pull requests, and contributors, which highlight their role in defining package popularity. Conversely, Maven-specific metrics, including dependencies and vulnerabilities, showed weak correlations with GitHub-based popularity indicators. Our analysis identified license status, commits count, presence of README files, and usages as the most significant predictors of package popularity, while vulnerabilities had limited statistical impact. These findings underscore the complementary nature of technical and community-driven metrics in assessing package popularity and provide actionable insights for developers and researchers to better evaluate and select open-source software packages.

Keywords: Popularity Metrics; Software Packages; Maven; GitHub

I. Introduction

The reusability of software packages (i.e., artifacts) has significantly enhanced developer productivity and improved the quality of software systems by enabling developers to build upon pre-existing packages. However, the extensive availability of open-source software packages in ecosystems, such as Maven[1], often makes it challenging for developers to select, contribute to, or reuse the most suitable packages. Popularity metrics, which quantify aspects of a package's usages, community engagement, and technical activity, play a crucial role in guiding these decisions.

This study investigates the role of popularity metrics in evaluating open-source software packages in Maven and addresses the following research questions: **(1)** How popular are the packages in the Maven ecosystem? **(2)** How are different popularity metrics of Maven packages correlated with each other? **(3)** Which of the studied metrics are the most important in determining highly popular packages?

To answer these questions, we analyzed 103,315 Maven packages, all of which are at least two years old. We collected metrics related to popularity from two key sources: the Maven Central Neo4j dataset [1] and GitHub[2] repositories. Metrics such as release frequency, dependencies, and usages reflect the technical and maintenance activity of packages, while GitHub metrics like star count, fork count, pull requests, contributors, and the presence of README files provide insights into community engagement and developer support [2]. Logistic regression analysis was employed to determine the impact of these metrics on the popularity of the Maven packages.

[1] https://central.sonatype.com/
[2] https://github.com/

 doi:10.20944/preprints202504.2296.v1

Our findings reveal that metrics such as license status, the number of commits, the presence of README files, and usages are the most significant predictors of package popularity, with strong statistical support. Notably, while vulnerabilities were intuitively expected to be impactful, they did not reach statistical significance in this analysis. This study provides actionable insights for developers and researchers, highlighting the most critical factors for evaluating and selecting open-source software packages effectively. For more details, please refer to the dataset and findings at **https://zenodo.org/records/14788435**.

The paper is organized as follows: Section II outlines data collection, including sources, metrics, and filtering. Section III presents the analysis on popularity, feature importance, and correlations. Section IV reviews related studies, and Section Vconcludes the paper.

II. Data Collection

The data collection process involves retrieving package IDs, release versions, the number of dependencies, usage counts, etc., from the Maven Central Neo4j dataset [1]; collecting GitHub source code URLs; gathering repository metrics from GitHub; and applying filtering techniques to ensure data quality and relevance.

From the Maven Central Neo4j dataset (Version 2024-08-30), we collect artifact IDs and the latest release versions. Using the information, we construct Maven URLs to retrieve `pom.xml` files for each package. If a GitHub repository URL is specified in the *pom.xml*, we collect the URL for further analysis.

We utilize the Goblin Weaver tool [3] to extract metrics from the Maven Central Neo4j dataset, encompassing release count, release frequency (i.e., how often new versions of a package are published within a given timeframe), dependencies (number of required packages), usages (number of packages that depend on it), popularity over the past year (activity level of the package), and vulnerabilities (count of vulnerable releases). Additionally, GitHub repository metrics—such as star count, fork count, pull requests, subscriber count, presence of a license (binary), tag count, open and closed issue counts, contributor count (number of individuals who contributed), commit count, creation and update timestamps, and the presence of README files (binary) and repository descriptions—are retrieved via the GitHub API. These metrics are then processed and structured for further analysis.

We collect 658,078 package IDs from the Maven Central Neo4j dataset. We remove 228,934 packages whose `pom.xml` files lack source code repository information. The remaining packages are distributed across 8,571 platforms, with the majority hosted on *https://github.com* (141,382 packages). For this study, we only consider those packages that keep their source code in GitHub repositories. We further remove 38,067 packages because their GitHub repositories are no longer available, leaving a final dataset of 103,315 packages for which we collect GitHub and Maven Central metrics.

III. Analysis and Result

This section addresses the research questions of our study.

RQ1: How Popular are the Packages in the Maven Ecosystem?

To evaluate the popularity of packages in the Maven ecosystem, we analyzed the distribution of packages based on their GitHub star count. GitHub stars serve as a widely accepted proxy for popularity, as they reflect user interest, adoption, and community endorsement. While not a perfect measure, stars indicate the visibility and perceived value of a package within the developer community [4]. Our analysis reveals a heavily skewed distribution, where most packages have low star counts, and only a small fraction achieve high levels of recognition.

Figure 1 illustrates the distribution of packages across different star ranges. The majority (i.e., 40%) of packages have star counts of 10 or less, indicating minimal popularity for a large portion of the dataset. As the star count increases, the number of packages decreases significantly, with only a small subset achieving widespread recognition.

Packages in the *1–10 stars* range dominate the dataset, with 44,388 packages representing repositories with minimal popularity. Moving to the *11–100 stars* range, 27,817 packages demonstrate moderate

 doi:10.20944/preprints202504.2296.v1

popularity, though still significantly below higher ranges. In the *101–1000 stars* range, 21,989 packages reflect a notable increase in recognition compared to the lower ranges. The *1001–10000 stars* range includes 10,282 highly popular repositories, while only 4,128 packages surpass the *10000+ stars* mark, making up the most widely recognized and utilized repositories.

These findings highlight the uneven distribution of package popularity in Maven, with a few gaining significant recognition while most receive little attention.

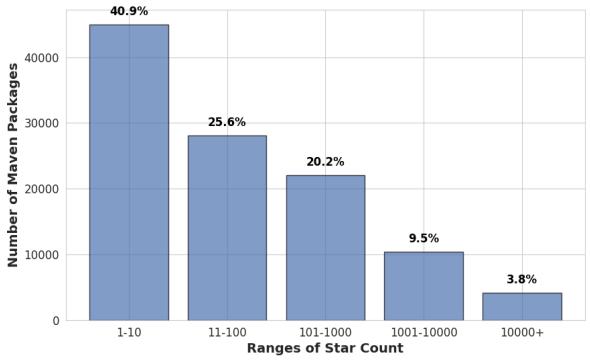

Figure 1. Distribution of Maven Packages by Star Count.

RQ2: How are Different Popularity Metrics of Maven Packages Correlated With Each Other?

To investigate the relationships between different popularity metrics (see Table 1) of Maven packages, we conducted a correlation analysis using both Pearson and Spearman methods. Pearson correlation measures the linear relationship between metrics, while Spearman correlation assesses their monotonic association, capturing both linear and non-linear trends. The results are presented in Figure 2, which provides a detailed view of the strength and direction of correlations among the metrics.

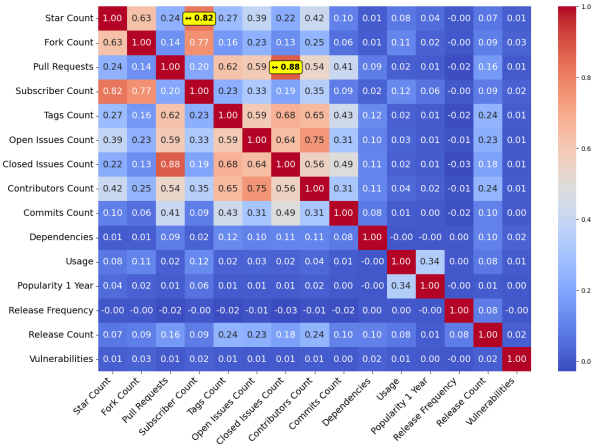

Figure 2. Correlation matrix of popularity metrics for Maven packages.

The analysis reveals that community-driven metrics, such as *stars* and *subscribers*, exhibit strong positive correlations (e.g., 0.82 between *stars* and *subscribers*). A similarly strong relationship is observed between *stars* and *forks* (0.63), indicating that popular repositories tend to attract more engagement in the form of forks and subscribers. Furthermore, *pull requests* show a strong correlation with *closed issues* (0.88), reflecting the active development and responsiveness of popular repositories.

Moderate correlations are observed between *contributors count* and *closed issues count* (0.56), as well as between *open issues count* and *tags count* (0.59). These findings indicate that repositories with more contributors and frequent updates handle issues more effectively, enhancing their popularity.

In contrast, technical metrics such as *dependencies* (both upstream and downstream), *release frequency*, and *vulnerabilities* exhibit weak or very weak correlations (less than 0.2) with community-driven metrics like *stars* and *forks*. For instance, the correlation between *dependencies* and other metrics

 doi:10.20944/preprints202504.2296.v1

is negligible, indicating that while dependencies may impact usability, they do not strongly correlate with engagement or popularity. Similarly, *vulnerabilities* show almost no relationship with metrics like *stars* or *forks*, implying that security concerns are not reflected in GitHub-based popularity indicators.

Overall, these findings highlight that community engagement metrics, such as *stars*, *subscribers*, and *forks*, are interrelated and serve as strong indicators of packages popularity. On the other hand, Maven-specific technical metrics, while critical for functionality, have limited influence on defining packages popularity and engagement.

RQ3: Which of the studied metrics are the most important in determining highly popular packages?

Table 1. Comparison of Features Across Top and Bottom 20% with p-value and Cohen's d for Effect Size; $d > 0.8$ is large (L), $0.4 < d < 0.8$ is medium (M), $0.2 < d < 0.4$ is small (S).

		Top 20%	**Bottom 20%**	
Feature	**Source**	**Mean/Median/Min/Max**	**Mean/Median/Min/Max**	***p*-value & Cohen's *d***
Star Count	GitHub	2162.74 / 415.0 / 31 / 229891	6.41 / 3.0 / 0 / 31	0.05 0.57 (M)
Fork Count	GitHub	418.42 / 53.0 / 0 / 47070	3.12 / 1.0 / 0 / 194	0.05 0.33 (S)
Pull Requests	GitHub	27.44 / 30.0 / 0 / 87	10.58 / 3.0 / 0 / 87	0.05 1.39 (L)
Subscriber Count	GitHub	60.99 / 18.0 / 0 / 6616	4.67 / 3.0 / 0 / 418	0.05 0.48 (M)
License	GitHub	———————	———————	———————
Tags Count	GitHub	23.56 / 30.0 / 0 / 30	10.35 / 5.0 / 0 / 30	0.05 1.26 (L)
Open Issues Count	GitHub	18.64 / 23.0 / 0 / 30	3.41 / 0.0 / 0 / 30	0.05 1.58 (L)
Closed Issues Count	GitHub	27.77 / 30.0 / 0 / 30	11.06 / 4.0 / 0 / 30	0.05 1.65 (L)
Contributors Count	GitHub	18.22 / 20.0 / 0 / 30	3.84 / 2.0 / 0 / 30	0.05 1.60 (L)
Commits Count	GitHub	29.66 / 30.0 / 0 / 30	25.57 / 30.0 / 0 / 30	0.05 0.65 (M)
README Exists	GitHub	———————	———————	———————
About Info	GitHub	5.59 / 5.0 / 1 / 71	4.33 / 4.0 / 0 / 47	0.05 0.31 (S)
Closed Issues Percentage	GitHub	0.64 / 0.52 / 0.0 / 1.0	0.55 / 0.70 / 0.0 / 1.0	0.05 0.26 (S)
Usages	Maven Neo4j	406.84 / 1.0 / 0 / 1716315	19.45 / 0.0 / 0 / 64281	0.05 0.05 (N)
Dependencies	Maven Neo4j	5.40 / 4.0 / 0 / 271	0.55 / 0.70 / 0.0 / 1.0	0.05 0.06 (N)
Popularity 1 Year	Maven Neo4j	9.39 / 0.0 / 0 / 89815	1.33 / 0.0 / 0 / 5516	0.05 0.02 (N)
Release Frequency	Maven Neo4j	0.09 / 0.02 / 0.0 / 122.4	0.11 / 0.01 / 0.0 / 10.0	0.05 -0.04 (N)
Release Count	Maven Neo4j	31.12 / 8.0 / 1 / 1288	9.09 / 3.0 / 1 / 691	0.05 0.36 (S)
Vulnerabilities	Maven Neo4j	0.01 / 0.0 / 0.0 / 20.5	0.00 / 0.0 / 0.0 / 165.0	0.05 0.00 (N)

Table 1 provides a comparative statistical summary of the metrics for the top 20% and bottom 20% of packages based on popularity. Metrics such as *Star Count* (Median: 415.0 for top 20% vs. 3.0 for bottom 20%) and *Fork Count* (Median: 53.0 for top 20% vs. 1.0 for bottom 20%) show substantial differences, with high statistical significance ($p < 0.05$) and medium to large effect sizes, underscoring their strong influence on package popularity. Similarly, metrics like *Closed Issues Count* and *Contributors Count* demonstrate large effect sizes, indicating their critical role in reflecting the activity and collaboration around the package.

Conversely, certain metrics, such as *Vulnerabilities*, *Usages*, and *Release Frequency*, showed negligible effect sizes, suggesting limited direct impact on package popularity. While these metrics provide valuable context about the package, their statistical insignificance highlights that developers may prioritize other factors, such as community engagement and active development, when assessing package adoption.

To identify the most significant metrics influencing the popularity of Maven packages, we performed a detailed quantitative analysis using logistic regression and hierarchical clustering. This analysis aimed to reduce multicollinearity among metrics and provide actionable insights into the factors most correlated with package popularity.

 doi:10.20944/preprints202504.2296.v1

A. Placeholder Subsection

1) Data Preprocessing and Feature Selection

The analysis considered packages that were at least two years old to ensure maturity and sufficient historical data. Metrics were standardized using standard scaling to normalize feature values across different ranges. The metric *About Info* was preprocessed by removing stop words and counting unique words to quantify the richness of the package descriptions.

Hierarchical clustering addressed multicollinearity, grouping metrics with a 0.7 cutoff. A representative metric was selected for each cluster based on interpretability and relevance.

The selected metrics include: *License, Commits Count, Readme Exists, About Info, Dependencies, Usages, Closed Issues Percentage, Release Frequency, and Vulnerabilities*.

2) Logistic Regression Methodology

Packages were categorized into “highly popular" (top 20% by star count) and “less popular" (bottom 20%) based on the star count of the packages. The middle 60% of packages were excluded to create a clear separation. A logistic regression model was trained using these labels as the dependent variable and the selected metrics as independent variables. The model’s performance was evaluated using the area under the ROC curve (AUC), with an AUC of 0.85 indicating strong discriminative ability.

3) Study Results

The logistic regression model results are summarized in Table 2, detailing the Wald χ^2-statistics, p-values, and significance levels for each metric.

Table 2. The results of our logistic regression analysis.

Metric	Wald χ^2	p-Value	Significance
License	1326.15	< 0.0001	***
Commits Count	1066.31	< 0.0001	***
README Exists	336.01	< 0.0001	***
About Info	658.34	< 0.0001	***
Dependencies	13.21	0.0003	***
Usages	475.63	< 0.0001	***
Closed Issues Percentage	9.26	0.0023	**
Release Frequency	11.00	0.0009	**
Vulnerabilities	3.39	0.066	.

Signif. codes: ‘***’ ($p < 0.001$), ‘**’ ($p < 0.01$), ‘.’ ($p > 0.05$).

The analysis reveals critical insights into the factors influencing package popularity in the Maven ecosystem. The most important factors include *License*, *Commits Count*, *README Exists* (i.e., the presence of README files), *About Info*, and *Usages*. Among these, *License* stands out as the most influential metric, with the highest Wald χ^2 value (1326.15). This result highlights the role of permissive licensing in encouraging broader usages and adoption of packages. Similarly, *Commits Count* is another vital metric, as frequent commits indicate active development and maintenance, which fosters user confidence in the package’s reliability. The presence of a *README* file is also significant, as it improves package accessibility by providing essential documentation, thereby making it easier for developers to adopt and integrate the package. *About Info* contributes to popularity by offering clear and detailed descriptions of the package’s functionality, enhancing its visibility and appeal. Finally, the *Usages* metric reflects the package’s real-world application and community adoption, serving as a direct indicator of its popularity.

Metrics such as *Closed Issues Percentage* and *Release Frequency* showed moderate importance. A higher percentage of closed issues signifies the maintainers’ ability to efficiently address problems, which enhances user satisfaction and confidence. Similarly, frequent releases suggest active main-

 doi:10.20944/preprints202504.2296.v1

tenance and responsiveness to evolving user needs, contributing to the package's reputation and reliability. These factors, while not as influential as the most critical metrics, still play an essential role in shaping user perception and adoption.

Interestingly, some metrics that are intuitively expected to be significant, such as *Vulnerabilities*, were found to have a limited statistical impact. Although a low number of vulnerabilities could enhance trust in a package, this metric did not achieve statistical significance ($p = 0.066$). This could be attributed to limited variation in vulnerabilities across the dataset or its indirect relationship with perceived popularity.

The analysis highlights that packages with permissive licenses, active development (high commits count), and robust documentation (README exists) are more likely to be highly popular. These insights can guide developers and ecosystem maintainers in prioritizing package features to enhance their utility and adoption.

IV. Related Work

Several prior studies investigated the popularity of open-source projects. Borges et al. investigated the use of stars in GitHub projects [5]. Their survey results showed that star count is considered the most useful measure for the popularity of GitHub projects by practitioners, followed by forks and watchers. In a separate study, they investigated the factors that affect the number of stars in GitHub repositories and identified four different growth patterns [4]. Prior studies also investigated the relation between the popularity of software projects and different factors. For example, Aggarwal found that consistent documentation update is a good indicator of project popularity [6]. Zhu et al. identified that popularity is related to the pattern of folder use [7]. Sajnani et al. investigated the relation between the popularity of projects and software quality (i.e., defect density and code quality) [8]. However, they did not find any relation. Wang et al. found that readme file-related factors can indicate the popularity of GitHub projects by analyzing 5000 GitHub projects across different languages [9]. Zerouali et al. showed that popularity can be measured in different ways but many popularity metrics are not correlated [10]. Similar to the prior studies, we also consider the star count as the popularity measure of the studied packages. We also investigate the correlation between different popular metrics. Despite using packages from a different ecosystem (i.e., Maven), our result is consistent with the study of Zerouali et al. Prior studies also determine the metrics that affect the popularity of GitHub projects [11] or highly selected packages [12]. Unlike others, we investigate Maven packages to determine the metrics related to highly popular packages.

V. Conclusion

This study analyzed metrics influencing Maven package popularity. Community-driven metrics like stars, forks, and pull requests strongly correlate with popularity, while Maven-specific metrics such as dependencies and vulnerabilities play a limited role. Key predictors include license status, commit count, README presence, and usage, emphasizing the value of documentation, active development, and community engagement. Interestingly, vulnerabilities had no significant impact. These findings provide actionable insights for evaluating open-source packages. Future work could explore temporal trends or other ecosystems for broader applicability.

References

1. D. Jaime, "Goblin: Neo4j maven central dependency graph," Sep 2024. [Online]. Available: https://doi.org/10.5281/zenodo.13734581
2. A. Abdellatif, Y. Zeng, M. Elshafei, E. Shihab, and W. Shang, "Simplifying the search of npm packages," *Information and Software Technology*, vol. 126, p. 106365, 2020.
3. D. Jaime, J. E. Haddad, and P. Poizat, "Navigating and exploring software dependency graphs using Goblin," in *Proceedings of the International Conference on Mining Software Repositories (MSR)*, 2025.

 doi:10.20944/preprints202504.2296.v1

4. H. Borges, A. Hora, and M. T. Valente, "Understanding the factors that impact the popularity of GitHub repositories," in *Proceedings of the IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2016, pp. 334–344.
5. H. Borges and M. T. Valente, "What's in a GitHub star? understanding repository starring practices in a social coding platform," *Journal of Systems and Software*, pp. 112–129, 2018.
6. K. Aggarwal, A. Hindle, and E. Stroulia, "Co-evolution of project documentation and popularity within GitHub," in *Proceedings of the ACM/IEEE Working Conference on Mining Software Repositories (MSR)*, 2014, pp. 360–363.
7. J. Zhu, M. Zhou, and A. Mockus, "Patterns of folder use and project popularity: A case study of GitHub repositories," in *Proceedings of the ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2014.
8. H. Sajnani, V. Saini, J. Ossher, and C. V. Lopes, "Is popularity a measure of quality? an analysis of Maven components," in *Proceedings of the IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2014, pp. 231–240.
9. T. Wang, S. Wang, and T.-H. P. Chen, "Study the correlation between the readme file of GitHub projects and their popularity," *J. Syst. Softw.*, Nov 2023.
10. A. Zerouali, T. Mens, G. Robles, and J. M. Gonzalez-Barahona, "On the diversity of software package popularity metrics: An empirical study of npm," in *Proceedings of the IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2019, pp. 589–593.
11. J. Han, S. Deng, X. Xia, D. Wang, and J. Yin, "Characterization and prediction of popular projects on GitHub," in *Proceedings of the IEEE Annual Computer Software and Applications Conference (COMPSAC)*, 2019, pp. 21–26.
12. S. Mujahid, R. Abdalkareem, and E. Shihab, "What are the characteristics of highly-selected packages? a case study on the npm ecosystem," *Journal of Systems and Software*, Apr 2023.