Correlation of Software Metrics and its Popularity

Tsung-Chieh Chen, Chen-Yu Yu, Kai-Chen Tung, Arvind Sudarshan, Anubhav Mishra

1. Introduction

1.1. Motivation

Conceiving the factors that contribute to the popularity of software projects holds significant importance for both individual developers and organizations [1]. For developers engaged in large-scale projects, knowing what makes a project successful can guide their efforts and increase the likelihood of their work being recognized and adopted by a broader community[1]. Additionally, the popularity of a project can serve as a source of motivation, as developers often derive satisfaction from seeing their creations appreciated and utilized by others [2].

For companies, the popularity of open-source projects can present valuable commercial opportunities. Projects with a large and active user base may attract investments, partnerships, or even acquisition offers[2]. Furthermore, contributing to popular open-source projects can enhance a company's reputation within the developer community, potentially leading to increased recruitment prospects and improved brand visibility [3].

Despite the prominence of languages like JavaScript in areas such as web development, other languages, such as Ruby, may not share the same level of widespread popularity[3] among their projects. Therefore, it is possible for a highly popular projects within less popular languages to be overlooked by the least popular projects of a more widely used programming language[4]. To mitigate this issue, our research takes a focused approach, examining projects within a single programming language and comparing the most and least popular GitHub repositories. This strategy aims to ensure fair comparisons, preventing the overshadowing of significant projects in less popular languages by those in more widely used languages.

1.2. Research Goal

The primary objective of this project is to investigate and analyze the diverse development practices that influence the popularity of software projects. Focusing on one of the most widely utilized programming languages, JavaScript[5], this study aims to conduct a comparative analysis to find association between highly popular and less popular JavaScript projects.

The popularity score for these repositories will be quantified using a composite metric that incorporates the number of stars, forks, and the square of pull requests [4]. This metric provides a comprehensive measure of a project's engagement level, reflecting both its attractiveness to users (stars and forks) and its collaborative activity (pull requests).

Our study aims to elucidate the impact of documentation, software quality, and developer diversity on the popularity of software development projects, specifically focusing on those utilizing JavaScript as their primary programming language within certain application domains. This targeted approach enables us to provide empirical evidence and actionable insights for developers, project managers, and stakeholders, highlighting practices that can enhance a project's visibility and success. By exploring associations and identifying patterns, we seek to equip developers and organizations with insights that inform their decision-making processes. Ultimately, gaining a deeper understanding of these dynamics can lead to more informed choices regarding project selection, resource allocation, and strategic planning, thereby fostering a more vibrant and inclusive software development community.

1.3. Research Questions

RQ1: Does the presence of documentation associate with popularity?

We explore the influence of documentation presence and quality on the popularity of open-source projects. Documentation, especially README files, plays a crucial role in a project's understanding, adoption, and contribution, often impacting its popularity [4, 5]. To the best of our knowledge, the specific impact of documentation within communities focused on a single programming language remains under explored. By analyzing documentation metrics, we aim to assess whether well-documented projects gain more attention and engagement from developers, thereby enhancing our understanding of documentation's significance in open-source development and informing practices to boost project visibility and success.

RQ2: Does software quality associate with popularity?

This research question delves into the relationship be-

[1] https://easy.bi/blog/project-success-factors/

[2] https://blog.logrocket.com/product-management/user-acquisition-strategies-mobile-apps-digital-products/

[3] https://octoverse.github.com/2022/top-programming-languages

[4] https://techwireasia.com/06/2023/top-modern-new-programming-languages-paving-way/

[5] https://github.blog/2023-11-08-the-state-of-open-source-and-ai/

tween software quality and the popularity of open-source projects. Understanding the link between software quality and project popularity is crucial for developers, project maintainers, and stakeholders within the software development community due to its wide-ranging implications. Previous studies, such as those by Alsmadi et al. [1], have highlighted a correlation between software complexity—a key aspect of software quality—and popularity. Motivated by these findings, our study seeks to further explore the potential connection between overall software quality and project popularity. Moving beyond previous research, we broaden our analysis to include a wider array of quality metrics, aiming for a comprehensive view of how various aspects of software quality influence a project's appeal. Our objective is to determine if projects characterized by higher quality levels are more likely to gain attention and engagement from the developer community. Through this research, we intend to uncover insights into the dynamics driving project popularity and offer guidance on enhancing software quality to boost project visibility and success within the open-source ecosystem. Ultimately, understanding how software quality affects project popularity is vital for refining development practices and maximizing the impact of projects among developers.

RQ3: What is the effect of diversity of developers on popularity?

We examine the impact of developer diversity on the popularity of open-source projects, underscoring its importance for promoting inclusivity, innovation, and collaboration in the software development ecosystem. Despite existing research on gender and tenure diversity [6], our study bridges the gap by associating developer diversity metrics, such as geolocation (time zone) and programming language diversity, with project popularity. We aim to understand whether projects with a diverse contributor base are more popular, offering insights into diversity's role in promoting innovation and community engagement, and shaping strategies for inclusivity and global collaboration in software teams.

1.4. Contributions

In this study, we make significant discovery in understanding the determinants of GitHub project popularity by addressing key gaps and providing actionable insights into roles of transparency, software quality, and diversity. All of these results in making this study have considerable contributions for future works and analyses, these are listed as follows:

- **Transparency and Governance Over Documentation**: We investigate the relationship between documentation, licensing, and project popularity. Contrary to existing assumptions, our analysis reveals that the presence of licensing and codes of conduct is more strongly correlated with popularity than documentation volume alone. This addresses the gap in previous research by emphasizing the crucial role of legal clarity and governance in fostering successful open-source communities.
- **Balancing Software Quality and Complexity** : Our study explores the impact of software quality metrics on project popularity. By examining bug proneness, code cleanliness, and cognitive complexity, we bridge a gap in understanding how these factors influence project success. Our findings underscore the importance of maintaining clean code and managing cognitive complexity effectively to enhance project appeal and popularity.
- **Embracing Diversity for Collaboration**: We investigate the effects of developer and programming language diversity on project popularity. This addresses a gap in the literature regarding the influence of diversity on project success. Our research highlights the positive impact of global contributor diversity while cautioning against excessive programming language diversity, providing valuable insights for fostering inclusive and successful open-source communities.

In summary, our research fills critical gaps in understanding GitHub project popularity, providing actionable recommendations for enhancing project success. By underscoring transparency, software quality, and diversity as crucial factors, we contribute valuable insights to this area of research.

For transparency and reproducibility, we have made our data and scripts available on GitHub[6]. The rest of the paper is structured as follows: Section 2 reviews related work, Section 3 describes our methodology, Section 4 presents our results, and Chapter 5 discusses the implications of our findings.

2. Background and Related Work

Previous research has explored various dimensions of open-source projects, including the influence of documentation [4, 7], software quality [8], gender, tenure [6] and diversity [9, 10] of the contributors. However these studies often focus on isolated aspects of popularity and have varying definitions of popularity. Moreover, the evolving nature of open-source communities and continuous developments in projects have inspired us to conduct an updated analysis that considers a broader range of factors and their interdependencies. Our study aims to bridge these gaps by adopting a comprehensive approach to analyze the development practices contributing to a project's popularity.

[6]https://github.com/tobyyu007/ECS260-Final-Project

2.1. Documentation as a measure of popularity

The study by Borges et al. [7] aimed to identify factors influencing the popularity of GitHub projects, focusing on metrics such as programming language, project domain, repository ownership, and documentation. It employed a mixed-methods approach, combining quantitative analysis with qualitative surveys. However, the study's reliance on stars as the primary measure of popularity was a limitation, potentially overlooking other significant factors.

Aggarwal et al. [4] explored the relationship between project documentation and popularity on GitHub. They utilized cross-correlation analysis to examine how documentation activity correlates with popularity metrics, including stars, forks, and pull requests. Their findings suggest a positive correlation between consistent documentation and higher project popularity.

Building upon these insights, our research aims to deepen the analysis by quantifying documentation's impact on popularity. We plan to evaluate additional metrics, such as the number of lines in markdown files, the total size of documentation files, and the frequency of documentation updates, LICENCES and CODE OF CONDUCT to provide a more comprehensive understanding of how documentation influences project popularity.

2.2. Role of software quality in project popularity

Kochhar et al. [11] investigate the impact of using multiple programming languages on software quality, particularly in terms of bug proneness. They examine whether there is a correlation between the number of programming languages used in a project and the number of bug fix commits, whether some programming languages are more bug-prone when used in combination with others, and the relationship between the number of languages used and the types of bugs encountered. Various factors such as project size, the number of developers, and commit history are considered to determine their correlation with bug proneness. Their findings indicate a positive correlation between the use of multiple languages and increased bug proneness.

Bogner et al. [8] provide valuable insights into software quality by examining aspects such as code smells, cognitive complexity, and bug resolution times in JavaScript (JS) and TypeScript (TS) projects. For their analysis, the authors collected and analyzed a large dataset from GitHub projects. They utilized SonarQube[7] and ESLint to measure code quality metrics, including code smells and cognitive complexity to mine data from GitHub repositories to compare JS and TS projects.

Kochhar et al. [11] and Bogner et al. [8] lay the groundwork in evaluating quality metrics within software projects. While their analyses offer valuable insights into the dimensions of software quality, a direct association with project popularity remains unexplored. Our research aims to delve into this potential relationship, examining how these quality metrics might influence a software project's attractiveness and engagement within the community. Through this approach, we aim to illuminate the impact of software quality on project popularity, thereby filling a notable gap in the existing literature.

2.3. Correlation between Diversity and project popularity

Xia et al. [9] investigate activities and contributors on GitHub with the aim of understanding the dynamics of who contributes, what their contributions entail, when these activities peak, and the geographic diversity of the contributors. Utilizing a dataset from the GitHub Event API, the paper tracks behaviors such as commenting on issues and PRs, opening issues, and merging PRs. The authors developed an activity model that combines these behaviors to pinpoint active contributors and deduce their working patterns. While the study provides extensive analysis of contributors' locations and active hours—shedding light on diversity in terms of time zones—it does not directly correlate these diversity aspects with project popularity. This represents a significant gap our research aims to fill by examining how these factors influence popularity.

Bissyande et al. [10] examine programming languages' popularity and interoperability in open-source projects on GitHub, focusing on their impact on project success. Their study of 100,000 projects reveals scripting languages' high interoperability and highlights that the size of the development team and the number of issue reports vary significantly across languages, without a clear correlation to the language used. While the study finds the correlation between the issues reported and the number of languages used, our research investigates the correlation between the diversity of programming languages in a project and its popularity.

3. Methodology

3.1. Study Design

Our study utilizes regression modeling to investigate the relationship between various predictors and the popularity score of software projects, specifically focusing on JavaScript applications. This includes applications in areas such as web, desktop, or smartphone applications, and excludes repositories related to frameworks like Vue.js[8] and educational resources such as freeCodeCamp[9]. This focus allows us to provide insights relevant to a significant segment of the open-source community.

[7] https://www.sonarsource.com/products/sonarqube/

[8] https://github.com/vuejs

[9] https://github.com/freeCodeCamp/freeCodeCamp

Inspired by previous research [12], we base our analysis on negative binomial regression (NBR), a generalized linear model ideal for non-negative integer responses. This choice is due to NBR's capacity to handle over-dispersion, where the variance of the response variable exceeds its mean [13].

In instances where the data distribution deviates from normality, transforming the data can help mitigate skewness and approximate a normal distribution more closely [14]. There are several transformation techniques available, including log transformation, inverse transformation, and Box-Cox transformation, etc. For our analysis, we apply the Box-Cox transformation for transforming the features, with the transformation defaulting to a log transformation when the lambda value is zero. The Box-Cox transformation is defined mathematically as follows:

$$y(\lambda) = \begin{cases} \frac{y^{\lambda-1}}{\lambda}, & \text{if } \lambda \neq 0. \\ \log y, & \text{if } \lambda = 0. \end{cases}$$

To ensure that excessive multi-collinearity does not compromise the integrity of our models, we calculate the variance inflation factor (VIF) for each predictor variable. While there is no universally accepted threshold for what constitutes excessive VIF, we follow the conservative threshold of 5 as a guideline [13]. This approach allows us to confirm the robustness of our regression models and the reliability of our findings concerning the factors influencing software project popularity.

3.2. Variables Considered

In the following subsections, we first discuss the variables considered for each RQ. After detailing the variables for each individual RQ, we will discuss the set of confounding variables applied across all RQs.

RQ1

For RQ1, our investigation targets documentation's influence on repository popularity, focusing on Markdown (`md`) and text (`txt`) files. We propose that thorough and updated documentation boosts a repository's appeal and user activity.

Number of documentation files: Comprehensive documentation is crucial for aiding developers and users in understanding, maintaining, and contributing to a repository [15]. We quantify this aspect by counting the documentation files with `md` (Markdown) and `txt` (text) extensions within a repository.

Number of changes to documentation files: As repositories evolve, their documentation must also be updated to inform users about changes, necessary adjustments, and new feature utilization [16]. We assess this dynamism by calculating the number of commits made to files identified as documentation.

Number of lines in documentation files: Detailed documentation, covering more aspects comprehensively, can significantly enhance user understanding of a repository [15]. We determine this by counting the lines in each identified documentation file, offering insights into the thoroughness of the information provided.

Size of documentation files: The number of lines might not always reflect the thoroughness of the documentation, since a document may contain many lines but fewer words, the size of the file provides a more direct measure of content volume. To assess this, we evaluate the size of documentation files in `bytes`.

Having a license or not: The presence of a clear and consistent license in a repository can be indicative of well-maintained and documented software, providing guidelines on the permissible uses, modifications, and distribution of the source code [17]. To determine the presence of a license, we examine whether a repository includes license keywords such as `MIT` or `Apache-2.0`.

Having a code of conduct or not: Code of conduct is crucial for defining engagement standards within a community, promoting an inclusive environment that respects all contributions, and detailing procedures for resolving conflicts among community members [18]. In line with GitHub's guidelines for establishing a code of conduct[10], we assess the presence of a `CODE_OF_CONDUCT` file in repositories.

RQ2

For RQ2, we delve into the relationship between software quality and the popularity of repositories by focusing on three indicators commonly used to assess software quality. We speculate that repositories with higher software quality are likely to be more popular, as they may provide a more reliable and user-friendly experience.

Code smells per LoC: Code smells are described as indicators of potential issues in software design or code that, while not directly causing bugs, suggest areas that may hinder future development efforts or increase the likelihood of bugs and failures [19]. Addressing and resolving code smells is essential for improving maintainability and preventing possible bugs [8]. To quantify the presence of code smells in a repository, we utilize the code analysis tool SonarQube[11]. To adjust for project size, which could affect the analysis, we measure code quality on a per-project basis as the number of code smells per line of code (LoC).

Cognitive Complexity per LoC: A key metric for

[10] https://docs.github.com/en/communities/setting-up-your-project-for-healthy-contributions/adding-a-code-of-conduct-to-your-project

[11] https://www.sonarsource.com/products/sonarqube/

measuring this aspect is cognitive complexity [20]. Developed as a human-centric understandability metric, cognitive complexity aims to overcome the limitations of cyclomatic complexity [21]. This metric has gained wide acceptance in the industry for providing effective evaluations of code comprehensibility [22]. To calculate cognitive complexity, we utilized SonarQube to assess cognitive complexity across repositories. Similar to code smells, we measure the code understandability of a project in terms of cognitive complexity per line of code (LoC).

Bug Proneness: Higher incidence of bugs can diminish user experience, identifying and classifying bugs accurately is crucial for maintaining the functional correctness of a repository, as unresolved bugs can lead to diminished user experience due to the software not functioning as expected [23]. To calculate bug proneness, we employed an issue classifier using the BART-large-mnli model[12] to analyze all commit messages, determining whether a commit addresses an issue or bug. Subsequently, we quantified bug proneness as the ratio of bug fix commits, that is, the number of commits addressing bugs divided by the total number of commits for a given project.

RQ3

For RQ3, we investigate the impact of developer diversity on the popularity of repositories. We hypothesize that repositories with greater diversity are likely to be more popular, owing to their inclusivity and the wide range of perspectives and functionalities they incorporate, which enrich the project's development and appeal.

Number of developers in different timezones: A diverse array of developers from various geographical locations can enhance a repository's popularity, as it reflects contributions from individuals with varied backgrounds, such as different educational institutions [24]. To assess geographical diversity, we calculate the number of distinct timezones represented in the commit messages of a repository.

Number of programming languages in a repository: Each programming language, with its unique characteristics, strengths, and weaknesses, contributes to the project's versatility and capability to address various functional domains effectively [25]. We determine the diversity of programming languages used in a repository by counting the distinct languages identified through GitHub's GraphQL API[13].

Confounding Variables

To mitigate the impact of confounding factors that could affect the outcomes of our RQs, we have identified three confounding variables applicable to all RQs.

Team Size: A larger team size can result in more contributors to the code base, potentially influencing analyses related to documentation, software quality, and repository diversity. Hence, team size is incorporated as a confounding variable in our study. We calculate team size for a repository by counting the number of contributors, as provided by the GitHub GraphQL API.

Repository Size (KB): The size of a repository might reflect a project with extensive documentation, substantial code, or a diverse development team, all of which could impact our analyses across RQs. We determine repository size by cloning the repository and measuring the total project size in Kilobytes (KB).

Repository Age: Older projects may have more documentation, largely due to their longer existence. Older project may also have a larger volume of code within a repository, which can influence evaluations of software quality. We determine the age of a repository by calculating the duration from its creation date to the time we accessed the repository, which was February 1st, 2023. For this calculation, we relied on the creation date from the GitHub GraphQL API.

3.3. Data collection

Our data pipeline encompasses three main steps: data gathering, filtering, and preprocessing. In the following subsections, we will provide details for each of these steps.

Data Gathering

We selected GitHub as our primary data source due to its vast repository of projects and advanced data extraction tools. According to a recent GitHub analysis[14], JavaScript emerged as the most popular programming language. Consequently, we selected JavaScript as our focus for analyzing repositories. Utilizing GitHub's GraphQL[15], we collected data on repositories created from GitHub's inception on 2008/01/01[16] until 2024/01/31. To circumvent GraphQL's query limitations (1000 repositories per query), we segmented our searches daily, maximizing our data collection efforts. Our search criteria included the keyword "javascript" and excluding forked repositories and focusing on the temporal aspect of the repository to analyze the repository at the current state, thereby gathering 1.8 million repositories.

[12] https://huggingface.co/AntoineMC/distilbart-mnli-github-issues?text=Remote+tmp+dir.++should+be+ignored+by+git

[13] https://docs.github.com/en/graphql

[14] https://github.blog/2023-11-08-the-state-of-open-source-and-ai/

[15] https://docs.github.com/en/graphql

[16] https://www.sitepoint.com/github-gist-is-pastie-on-steroids/

Data Filtering

To align with our research objectives, we refined our dataset to include only JavaScript repositories within the application domain. This was achieved using Latent Dirichlet Allocation (LDA)[17] to analyze README files, identifying topics represented by a set of words and assigning documents to these topics with a probability score. Following the methodology inspired by [8], we identified 30 domains and excluded topics such as "plugin," "module," "extension," "API," "database," "framework," and "library." After domain filtering, we then retain active repositories by setting thresholds for stars (≥ 2), commits (≥ 20), and pull requests (≥ 9). We used the following formula [4]:

$$\text{Popularity Score} = \#\text{Stars} + \#\text{Forks} + (\#\text{Pull Requests})^2$$

to define popularity. We then select the top 60 popular and bottom 60 unpopular repositories for further analysis.

For RQ1, identifying relevant documentation involved automated filtering to pinpoint files with `md` (Markdown) and `txt` (text) extensions, excluding files under three words and those serving a special purpose (e.g., `requirements.txt`), but retaining `robots.txt` due to its website indexing role. Despite automation, further refinement was necessary to enhance analysis quality. We conducted a manual review for removing non-relevant documentation based on criteria such as files containing backup code, outdated documents, or merely file with names.

Data preprocessing

In the data preprocessing phase, we extracted and processed specific attributes from the filtered GitHub repositories. After acquiring the essential attributes mentioned from section 3.2, we organized this information within a MongoDB database to facilitate team collaboration. This approach allowed the data to be shared among team members.

In addressing RQ1, we leveraged various Python libraries, including os and Lizard, to traverse each repository's directories and identify documentation files. This enabled us to systematically count and aggregate both the size and line count of these files. Additionally, we employed PyDriller to iterate through all commits affecting documentation files, which facilitated the quantification and accumulation of the number of changes made to the documentation files.

For RQ2, we focused on gathering data related to code quality. This involved collecting metrics on code smells and cognitive complexity for each repository, using SonarQube for the calculations. To normalize these metrics, we divided them by the number of lines of code. For assessing bug proneness, we analyzed commit messages with an issue classifier trained on the BART-large-mnli model[18], which predicts whether a commit message classified to a bug. We then computed the proportion of commits identified as addressing bugs relative to the total number of commits, thereby obtaining a measure of bug proneness for each repository.

In data collection for RQ3, we meticulously examined all commits within each repository to ascertain the distinct timezones of the committers. Leveraging the timezone information provided by PyDriller, we computed both the total number of distinct timezones represented and the count of committers within each timezone for every repository. Additionally, to determine the number of programming languages utilized within a repository, we inspected the 'languages' attribute sourced from GitHub's GraphQL API.

This approach in both data gathering and filtering phases ensures our analysis is grounded in relevant and accurately categorized repositories, paving the way for insightful conclusions on the relationship between documentation, software quality, developer diversity, and repository popularity.

3.4. Analysis Methods

In the subsections below, we detail our analysis methods, presenting findings for each RQ in dedicated subsections.

RQ1

Before progressing to regression analysis, we conducted a descriptive analysis to familiarize ourselves with the data. Utilizing box plots and histograms, we visualized the dataset's distribution, providing initial insights relevant to our RQ.

Following this initial review, we utilized a correlation matrix to assess the collinearity among the attributes, ensuring that the Variance Inflation Factor (VIF) did not exceed the widely accepted benchmark of 5 [13]. Additionally, we addressed the challenge of non-normally distributed data by implementing appropriate transformations. We applied a Box-Cox transformation to several variables, including repository size, repository age, team size, and the count of documentation files. The Q-Q plot to the left in Figure 1 displays the initial skewness in the number of documentation files, while the plot to the right, after Box-Cox transformation, indicates a distribution that more closely resembles a normal distribution. We also applied transformation for the categorical variables "has license" and "has code of conduct" were transformed into numerical values, 0 for False and 1 for True to facilitate their integration into the regression model.

We then proceeded with regression analysis using a Neg-

[17] https://radimrehurek.com/gensim/models/ldamodel.html

[18] https://huggingface.co/AntoineMC/distilbart-mnli-github-issues?text=Remote+tmp+dir.++should+be+ignored+by+git

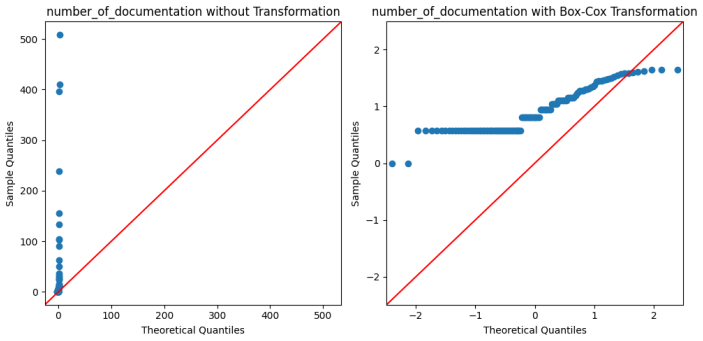

Fig. 1: Q-Q Plot for Number of Documentation before (Left) and after Box-Cox transformation (Right)

ative Binomial Distribution, aiming to interpret the results and their potential implications for open-source software repositories. We examined the p-value with a standard threshold of 0.05 from the regression model to decide on accepting or rejecting the null hypothesis, allowing us to understand of how documentation or licensing correlates with project popularity.

RQ2

Adopting the methodological approach from RQ1, we began examining RQ2 with a descriptive analysis to acquire a foundational understanding of our data before conducting regression analysis. Box plots and histograms provided visualization of the dataset's distribution and provided preliminary insights into this research question.

After this preliminary analysis, we proceeded to conduct regression analysis using a Negative Binomial Distribution. As a preparatory step for the regression analysis, we employed a correlation matrix to assess collinearity among the attributes. We also calculated the VIF to confirm that it remained below the widely recognized threshold of 5 [13]. In this RQ, we applied a Box-Cox transformation to several variables, including repository size, repository age, team size, and the count of documentation files.

After completing the regression analysis, we will interpret the results to understand their impact in open-source software projects. By evaluating the p-values with a standard threshold of 0.05 obtained from the regression model, we will make informed decisions about accepting or rejecting the null hypothesis.

RQ3

For RQ3, we also conducted a descriptive analysis to deepen our understanding of the data before proceeding to regression analysis. The use of box plots and histograms enabled us to both visualize the dataset's distribution and obtain early insights relevant to our RQ.

Following this initial analysis, we applied regression analysis by employing a Negative Binomial Distribution. We use a correlation matrix to check for collinearity among attributes and verifying that the VIF remained below the established threshold of 5 [13]. We utilized for a Box-Cox transformation on variables such as repository size, repository age, team size, the number of developers across different timezones, and the diversity of programming languages used in the repository.

We will then interpret the regression analysis results to evaluate their significance in the context of open-source software projects. Utilizing the standard p-value threshold of 0.05 from the regression model, we will determine whether to accept or reject the null hypothesis.

4. Results

RQ1: Does the presence of documentation or license associate with popularity?

RQ1 investigates whether the presence of documentation is correlated with a repository's popularity on GitHub, taking into account control variables such as team size, repository size, and repository age. Following the analytical method outlined in Section 3.4, we initially conducted a descriptive statistics, followed by correlation analysis, and then regression analysis.

Table I presents the descriptive statistics, underscoring a stark contrast in the number of documentation files between popular and unpopular GitHub repositories. Our findings reveal that all popular repositories contain at least one piece of documentation, while some unpopular ones lack any documentation altogether. Popular repositories, on average, host considerably more documentation files (40.48) than unpopular ones (5.85), a trend that remains consistent when considering median values. The maximum values reveal that the most documented popular project has 509 pieces of documentation, which far exceeds the maximum in unpopular projects (238). This suggests that a higher number of documentation files might be indicative of more successful projects.

The result of our correlation analysis, depicted in Figure 2, revealed significant collinearity among documentation-related attributes. Consequently, we opted to include only the 'Number of Documentations' variable in our analysis, excluding other related metrics to preserve analytical integrity. To further ensure minimal collinearity among the selected variables, we conducted VIF checks. Following the analysis method provided in section 3.4, our metrics met the recommended VIF limit of 5, as shown in Table II.

Results from our Negative Binomial Regression, as outlined in Table II, show that although the quantity of documentation is not significantly linked to popularity (with a p-value of 0.294), the presence of licensing and codes of conduct in repositories is significantly positively correlated

	RQ1	RQ2			RQ3	
	Number of Documentation	Bug Proneness	Code Smell per LoC	Cognitive Complexity per LoC	Number of Different Timezones	Number of Programming Languages Used
Min	(1, 0)	(0.0036, 0)	(0, 0)	(0, 0)	(1, 1)	(1, 1)
Mean	(40.48, 5.85)	(0.13, 0.11)	(0.04, 0.04)	(0.11, 0.06)	(8.07, 2.05)	(4.40, 2.85)
Median	(6, 1)	(0.12, 0.07)	(0.03, 0.02)	(0.06, 0.03)	(5.50, 1)	(4, 3)
Max	(509, 238)	(0.76, 0.64)	(0.20, 0.18)	(0.94, 0.4)	(26, 10)	(10, 5)

TABLE I: Descriptive analysis for RQ1, RQ2, and RQ3. The figures in parentheses indicate the results for popular and unpopular repositories, respectively (e.g., (popular, unpopular))

Metrics	VIF	Coefficient	Standard Error	p-Value	Accepted
(Intercept)	-	-0.27	0.697	0.695	-
Team Size	-	0.62	0.157	0.000	Yes
Repository Size	-	0.57	0.075	0.000	Yes
Repository Age	-	0.29	0.032	0.000	Yes
Number of Documentations	1.49	0.37	0.355	0.294	No
Has License	1.24	0.74	0.222	0.001	Yes
Has Code of Conduct	1.37	2.05	0.288	0.000	Yes

TABLE II: Regression analysis results for RQ1

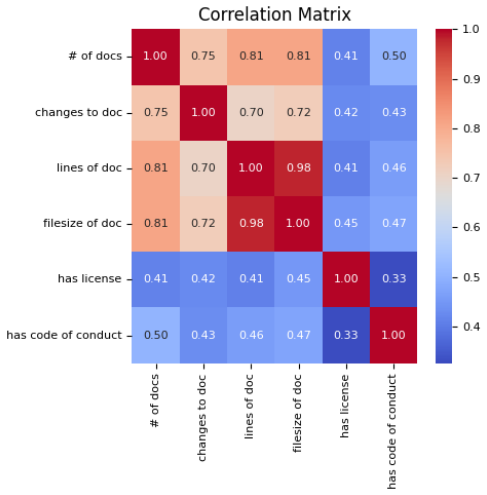

Fig. 2: Correlation Matrix for RQ1

with popularity ($p \leq 0.001$). These findings underscore the significance of legal clarity and structured governance in boosting a project's attractiveness. Noteworthy is that confounding factors like repository age, size, and team size also exhibit a positive correlation with popularity, suggesting that maturity, substantial resources, and a larger collaborative team are beneficial traits for open-source projects.

While the descriptive analysis initially suggested a potential link between the volume of documentation and popularity, the regression analysis introduced a nuanced view, suggesting that increased documentation alone may not directly lead to higher popularity. This nuance hints at the possibility that an abundance of documentation could result from enhanced contributor activity, rather than being a causal factor for popularity.

> **Result 1:** A license and code of conduct within a GitHub repository are more strongly linked to popularity than the sheer volume of documentation.

RQ2: Does software quality associate with popularity?

The aim of RQ2 is to explore the impact of software quality on the popularity of GitHub projects, specifically through metrics such as bug proneness, code smells per line of code, and cognitive complexity per line of code. We account for confounding factors like repository age, size, and team size to ensure a comprehensive analysis. Our approach started with a correlation analysis to identify potential relationships among variables. This was followed by descriptive statistics to summarize the characteristics of our dataset. Finally, we conducted regression analysis to quantify the influence of software quality on project popularity.

The descriptive statistics (Table I) for RQ2 offer insights on differences between popular and unpopular repositories concerning software quality. Notably, popular repositories show a marginally higher mean of bug proneness (0.13) compared to unpopular ones (0.11), but the differences are slight. Both groups have similar averages for code

Metrics	VIF	Coefficient	Standard Error	p-Value	Accepted
(Intercept)	-	-3.52	0.678	0.000	-
Team Size	-	0.56	0.148	0.000	Yes
Repository Size	-	0.87	0.066	0.000	Yes
Repository Age	-	0.43	0.031	0.000	Yes
Code Smells per LoC	1.97	0.050	0.050	0.036	Yes
Cognitive Complexity per LoC	1.69	0.050	0.050	0.000	Yes
Bug Proneness	1.41	0.050	0.050	0.838	Yes

TABLE III: Regression analysis results for RQ2

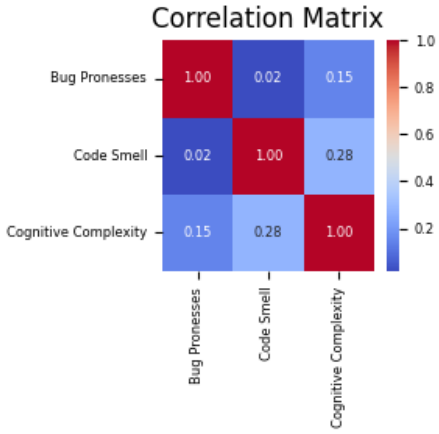

Fig. 3: Correlation Matrix for RQ2

smells per line of code (0.04), though popular repositories slightly edge out in medians (0.03 vs. 0.02). Cognitive complexity per line of code presents a clearer distinction; popular repositories have a higher average (0.11) compared to unpopular ones (0.06), indicating more complex code in popular projects. The maximum values observed for bug proneness, code smell, and cognitive complexity are higher for popular repositories, indicating that while popular projects may have more complex code, they do not necessarily have a higher incidence of bugs or code smells. These findings provide a preliminary insights for further regression analysis on the impact of software quality on project popularity.

Our correlation matrix analysis and VIF checks revealed no significant collinearity among the selected variables for RQ2, as demonstrated in Figure 3 and Table III, with all VIF values well below the threshold of 5, suggesting our model variables are not collinear.

The regression analysis findings, detailed in Table III, highlights several key findings. Bug proneness does not significantly influence popularity ($p = 0.838$), suggesting it might not be a primary concern for repository users. In contrast, a significant negative correlation between code smell per line of code and popularity ($p = 0.036$) highlights that cleaner code might attract more users. Surprisingly, a significant positive correlation between cognitive complexity per line of code and popularity ($p < 0.001$) suggests that more complex projects might be more appealing, possibly due to their advanced features. Both repository age and size show significant positive associations with popularity ($p < 0.001$), underscoring the advantages of established, resource-rich projects tend to attract attention. Similarly, a larger team size positively impacts popularity ($p < 0.001$), emphasizing the value of collaboration in achieving popularity.

These findings illustrate the complex nature of project popularity, where minimizing code smells appears more beneficial than merely reducing bugs. They indicate that while complexity is inevitable in some popular projects, it should be carefully managed to keep users engaged. Our analysis suggests active management of code smells and consideration of cognitive complexity relative to project goals as strategies for enhancing project appeal and user engagement.

Result 2: Maintaining clean code and managing cognitive complexity effectively are crucial for enhancing the popularity of GitHub projects.

RQ3: Does diversity of developers and different programming languages used associate with popularity?

In RQ3, we investigate the influence of developer diversity and the variety of programming languages used on the popularity of GitHub projects. Our analysis began with a correlation assessment, proceeded with descriptive statistics, and concluded with regression analysis, following the approach taken in RQ1 and RQ2.

Descriptive statistics, presented in Table I, reveal that popular repositories tend to involve contributors from a broader range of time zones, with a mean of 8.07 compared to 2.05 for less popular ones, indicating extensive global collaboration. This is further emphasized by the higher maximum of different time zones (26 for popular versus 10 for unpopular). When it comes to programming language

Metrics	VIF	Coefficient	Standard Error	p-Value	Accepted
(Intercept)	-	1.17	0.780	0.133	-
Team Size	-	0.50	0.151	0.001	Yes
Repository Size	-	0.77	0.075	0.000	Yes
Repository Age	-	-0.03	0.041	0.447	Yes
Number of Timezone	2.34	0.050	0.208	0.000	Yes
Number of Programming Languages	2.34	-0.34	0.162	0.035	Yes

TABLE IV: Regression analysis results for RQ3

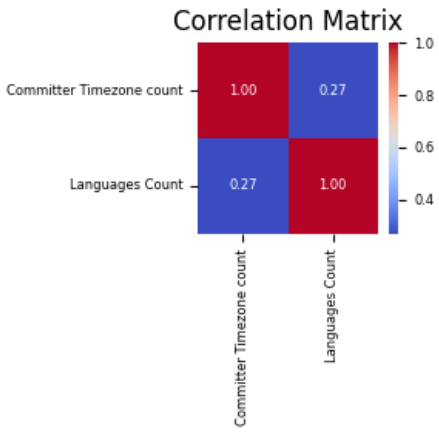

Fig. 4: Correlation Matrix for RQ3

diversity, popular repositories again show a slightly higher mean (4.40 against 2.85 for unpopular), with the most popular repositories having up to 10 different languages compared to 5 in the less popular ones. This suggests that not only the embrace of diverse programming languages but also the inclusion of contributors from numerous time zones may be factors that contribute to a repository's popularity.

Our correlation analysis (Figure 4) demonstrated low collinearity among the diversity metrics, confirming the distinct contribution of each to understanding a project's characteristics. The VIF results, detailed in Table IV, further shows no concerning levels of collinearity that would compromise the model's integrity, with all VIF scores below the threshold of 5.

The regression analysis underscored the positive impact of geographical diversity among contributors on project popularity ($p < 0.001$), advocating for the benefits of a globally distributed team. Conversely, an extensive use of programming languages within a project was found to negatively impacted popularity ($p = 0.035$), possibly due to the complexities and coordination challenges it introduces. Notably, factors such as repository age, size, and team size were all positively correlated with popularity (all having $p < 0.001$, except repository age), suggesting that well-established projects with substantial resources and larger collaborative teams are more likely to attract attention. These results show that worldwide collaboration boosts a project's popularity, but suggest not using to much programming languages in a project.

Result 3: Global contributor diversity positively influences GitHub project popularity, while excessive programming language diversity may introduce challenges that slightly reduce popularity.

5. Discussion

5.1. Main Findings

RQ1 reveals that GitHub project popularity involves more than just documentation. Despite expectations, our findings show that legal assurances and coding guidelines, indicated by the presence of a license and code of conduct, significantly impact popularity. This highlights the community's valuation of legal clarity and governance over sheer volume of documentation. The results suggest that for open-source projects to thrive, maintainers should prioritize establishing transparent licensing and comprehensive community guidelines. These measures are crucial in fostering a safe, inclusive, and well-regulated collaborative environment, underpinning the strategic development of successful open-source projects.

RQ2 analysis highlights the nuanced relationship between software quality and GitHub project popularity. The minimal effect of bug proneness challenges the idea that software must be flawless to be popular. The negative correlation with code smells underscores a preference for clean coding over quick bug fixes, suggesting foundational quality matters more to potential contributors. The attraction to cognitive complexity reflects an appreciation for sophisticated projects, indicating a desire for challenging, yet manageable, contributions. This balance between innovation and accessibility is crucial for fostering an inclusive and engaged community are key to a project's success on GitHub.

RQ3 explored the effects of developer diversity and the use of different programming languages on project

popularity. The results demonstrated that global contributor diversity significantly enhances project popularity, reflecting the value of wide-reaching collaboration. Conversely, an extensive array of programming languages within a project was found to slightly reduce popularity, suggesting the need for careful technological strategy to ensure project clarity and ease of contribution. The positive influence of global developer diversity on popularity suggests that projects benefit from inclusive and wide-reaching collaboration. Stakeholders should leverage tools and platforms that facilitate global contribution, emphasizing the creation of a welcoming and inclusive community. Conversely, the slight negative impact of extensive programming language diversity on popularity cautions against the potential complications of managing a multilingual codebase. Stakeholders should consider carefully selecting a coherent set of programming languages that serve the project's needs without introducing unnecessary complexity.

5.2. Strengths and Limitations

Strengths

Our study's targeted approach on JavaScript, one of the most utilized programming languages, stands out as a crucial strength. By narrowing our focus, we not only enhance the applicability of our findings to a wide developer audience but also ensure that significant projects in less prevalent languages are not overlooked. This specificity guarantees our insights are both relevant and actionable, addressing a common research oversight.

The adoption of a comprehensive analytical framework, integrating descriptive statistics, correlation analysis, and regression modeling, marks another strength. By controlling for various confounding variables, we refine the precision of our analysis. This approach not only enables us to derive meaningful insights but also ensures a solid foundation for our study. The isolation of primary variables adds an extra layer of accuracy to our findings.

Lastly, the diversity of our dataset, covering aspects like documentation, software quality, and developer diversity, provides a rich basis for analysis. By applying Box-Cox transformation and conducting VIF checks for multicollinearity, along with employing Negative Binomial Distribution for regression analysis, we ensure the robustness and practical applicability of our results. These methodologies greatly enhance the reliability and validity of our regression models, offering a holistic view of what contributes to the success of GitHub projects.

Limitations

Our study acknowledges certain limitations from the viewpoints of construct, internal, external validity, and empirical reliability, as detailed below:

Construct Validity: A notable limitation is our analysis's potential omission of documentation housed on wikis or external websites. This limitation could undermine construct validity by omitting comprehensive documentation sources, potentially affecting the accuracy of our findings.

Internal Validity: The presence of potential confounding variables, such as team size, poses a threat to internal validity. Larger teams might lead to more extensive and frequent updates to documentation, thereby influencing a repository's perceived popularity. To counter this, we incorporated team size as a control variable in our regression analysis, aiming to isolate the impact of documentation on popularity and mitigate the confounding effects of team size.

External Validity: Our exclusive reliance on GitHub data might not encapsulate behaviors observable on other platforms such as SourceForge or GitLab. Moreover, our focus on application domain repositories may limit the broader applicability of our findings to other domains with distinct dynamics and documentation requirements.

Empirical Reliability: The dataset's retrieval date of January 30, 2024, presents a limitation concerning empirical reliability. Changes in a repository's documentation or viability post this date could affect the study's replicability. Future research should consider the timing of data extraction and changes in repository visibility to ensure accurate replication.

5.3. Future Work

While our study encompasses a broad range of data and analysis methods, time constraints have necessitated the identification of potential avenues for future research. Regarding RQ1, an in-depth examination of documentation quality and relevance is recommended to elucidate their impacts on open-source project popularity. For RQ2, future studies could further explore complexity in popular projects to provide more concrete guidance for developers on balancing sophistication with user accessibility. RQ3's initial findings encourage further investigation into the strategic selection of programming languages to enhance a project's attractiveness and success in the open-source domain.

To extend the breadth and depth of our analysis, incorporating a diverse array of programming languages beyond JavaScript could yield more comprehensive insights, comparing development practices and their influence on project popularity across varied programming contexts. Moreover, a longitudinal study approach, tracing the development and evolution of projects over time, would shed light on the dynamic interplay between software quality, documentation, and developer diversity on project popularity, uncovering emergent trends and shifts. Additionally, integrating user feedback, including issue comments and project ratings, into our analysis framework promises to enrich our under-

standing of project popularity by weaving in user perceptions alongside the technical and social metrics previously explored. These research paths not only stands to deepen our grasp of what drives project popularity but also expands the research horizon within the open-source ecosystem, contributing valuable insights to developers, project managers, and the broader community.

6. Team membership and attestation

Team members Tsung-Chieh Chen, Chen-Yu Yu, Kai-Chen Tung, Arvind Sudarshan, and Anubhav Mishra participated sufficiently in the project.

References

[1] Izzat Alsmadi and Iyad Alazzam. "Software attributes that impact popularity". In: *2017 8Th international conference on information technology (ICIT)*. IEEE. 2017, pp. 205–208.

[2] Felipe Fronchetti et al. "What attracts newcomers to onboard on oss projects? tl; dr: Popularity". In: *Open Source Systems: 15th IFIP WG 2.13 International Conference, OSS 2019, Montreal, QC, Canada, May 26–27, 2019, Proceedings 15*. Springer. 2019, pp. 91–103.

[3] Carlos Santos et al. "The attraction of contributors in free and open source software projects". In: *The Journal of Strategic Information Systems* 22.1 (2013), pp. 26–45.

[4] Karan Aggarwal, Abram Hindle, and Eleni Stroulia. "Co-evolution of project documentation and popularity within github". In: *Proceedings of the 11th working conference on mining software repositories*. 2014, pp. 360–363.

[5] Akhila Sri Manasa Venigalla and Sridhar Chimalakonda. "An Empirical Study On Correlation between Readme Content and Project Popularity". In: *arXiv preprint arXiv:2206.10772* (2022).

[6] Bogdan Vasilescu et al. "Gender and tenure diversity in GitHub teams". In: *Proceedings of the 33rd annual ACM conference on human factors in computing systems*. 2015, pp. 3789–3798.

[7] Hudson Borges, Andre Hora, and Marco Tulio Valente. "Understanding the factors that impact the popularity of GitHub repositories". In: *2016 IEEE international conference on software maintenance and evolution (ICSME)*. IEEE. 2016, pp. 334–344.

[8] Justus Bogner and Manuel Merkel. "To type or not to type? a systematic comparison of the software quality of javascript and typescript applications on github". In: *Proceedings of the 19th International Conference on Mining Software Repositories*. 2022, pp. 658–669.

[9] Xiaoya Xia et al. "Exploring activity and contributors on GitHub: Who, what, when, and where". In: *2022 29th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE. 2022, pp. 11–20.

[10] Tegawendé F Bissyandé et al. "Popularity, interoperability, and impact of programming languages in 100,000 open source projects". In: *2013 IEEE 37th annual computer software and applications conference*. IEEE. 2013, pp. 303–312.

[11] Ana Trisovic et al. "A large-scale study on research code quality and execution". In: *Scientific Data* 9.1 (2022), p. 60.

[12] Baishakhi Ray et al. "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*. 2014, pp. 155–165.

[13] Jacob Cohen et al. *Applied multiple regression/correlation analysis for the behavioral sciences*. Routledge, 2013.

[14] Shovan Chowdhury et al. "Evaluation of tree based regression over multiple linear regression for non-normally distributed data in battery performance". In: *2022 International Conference on Intelligent Data Science Technologies and Applications (IDSTA)*. IEEE. 2022, pp. 17–25.

[15] Akhila Sri Manasa Venigalla and Sridhar Chimalakonda. "What's in a GitHub Repository?–A Software Documentation Perspective". In: *arXiv preprint arXiv:2102.12727* (2021).

[16] Akhila Sri Manasa Venigalla and Sridhar Chimalakonda. "Understanding emotions of developer community towards software documentation". In: *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS)*. IEEE. 2021, pp. 87–91.

[17] Thomas Wolter et al. "Open source license inconsistencies on github". In: *ACM Transactions on Software Engineering and Methodology* 32.5 (2023), pp. 1–23.

[18] Renee Li et al. "Code of conduct conversations in open source software projects on github". In: *Proceedings of the ACM on Human-computer Interaction* 5.CSCW1 (2021), pp. 1–31.

[19] Mohammed Lafi et al. "Code smells analysis mechanisms, detection issues, and effect on software maintainability". In: *2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT)*. IEEE. 2019, pp. 663–666.

[20] Marvin Muñoz Barón, Marvin Wyrich, and Stefan Wagner. "An empirical validation of cognitive complexity as a measure of source code understandability". In: *Proceedings of the 14th ACM/IEEE international symposium on empirical software engineering and measurement (ESEM)*. 2020, pp. 1–12.

[21] G Ann Campbell. “Cognitive complexity: An overview and evaluation”. In: *Proceedings of the 2018 international conference on technical debt*. 2018, pp. 57–58.

[22] Rubén Saborido et al. “Automatizing software cognitive complexity reduction”. In: *IEEE Access* 10 (2022), pp. 11642–11656.

[23] Wei Zheng et al. “A comparative study of class rebalancing methods for security bug report classification”. In: *IEEE Transactions on Reliability* 70.4 (2021), pp. 1658–1670.

[24] Davide Rossi and Stefano Zacchiroli. “Geographic diversity in public code contributions: an exploratory large-scale study over 50 years”. In: *Proceedings of the 19th International Conference on Mining Software Repositories*. 2022, pp. 80–85.

[25] Wen Li et al. “Understanding language selection in multi-language software projects on GitHub”. In: *2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE. 2021, pp. 256–257.