Exploring the Association of Project Popularity on Code Coverage: Insights from Popular Repositories

Raj Kunamaneni, Henry Chou, Anant Vishwakarma, Georgy Zaets, Srivatsan Srikanth

***Abstract*—GitHub hosts a diverse range of software projects, with popularity often measured by metrics like stars, forks, and pull requests. Despite this, the relationship between popularity and code coverage remains unexplored. To the best of our knowledge, we are the first to analyze the relationship between project popularity and code coverage metrics. This study addresses this gap by investigating the association between project popularity and code coverage metrics. Utilizing tools such as *Codecov* and *Coverall*, we gather code coverage reports and perform regression analysis to discern patterns in the data. Our findings reveal a significant correlation between project popularity and code coverage, generally showing an increase in coverage with popularity. However, while high code coverage moderately correlates with popularity, factors like pull requests and forks also influence project success. Notably, the introduction of new features, measured by pull requests, positively correlates with repository popularity. These results emphasize the multidimensional nature of project evaluation on GitHub, necessitating the integration of various metrics for a comprehensive understanding of project success.**

***Index Terms*—github, code quality, code coverage, project popularity, software development.**

I. Introduction and Goals

GitHub stands out as a leading platform for hosting a wide array of software projects, ranging from small-scale initiatives to large-scale endeavors[1]. It is extensively utilized by software development teams, highlighting the importance of understanding whether their projects on GitHub are effectively attracting and retaining users and contributors [1]. As a platform that rapidly grew in usage, GitHub has become a vital necessity for developers aiming to enhance their project's visibility [2]. The platform's greater transparency of actions undertaken within software development [3], has paved the way for a multitude of studies pertaining to software quality and productivity [4], [5].

Beyond mere technicalities, maintaining good software quality has also been shown to be an equally vital business concern, that needs to deserve the same attention as other measures of the financial success of a business [6]. This notion is further reinforced by the fact that popular repositories typically correlate with having more comprehensive project documentation [7], suggesting a link between popularity and project quality. Despite the sheer influence that design patterns exert in the field of software engineering [8], made famous by the "Gang of Four" textbook on the patterns of software design [9], in reality, using them is not guaranteed to improve code quality, and could even worsen the existing quality of code [10]. This leaves code coverage as one of the key measures of code quality [11], [12], [13] that has led to better reliability and lower error rates in software development [11], [13].

[1]https://github.com/features

However, to the best of our knowledge, we are the first to analyze the relationship between project popularity and code coverage metrics. None of the existing studies on popularity have considered code coverage levels as one possible variable that could correlate with changes in repository popularity levels. In fact, existing studies on the relationship between code quality and popularity have been extremely limited in scope, focusing only on very specific areas or domains. Sajnani *et al.* looked at the relationship between software quality and popularity [14], finding that no significant correlations existed between the two aspects, merely from the analysis of two narrowly scoped datasets: *Maven* software artifacts written in *Java* and *Java* projects from the *Sourcerer* repository. Capra *et al.* discovered that firm involvement in open source projects did indeed increase popularity at the cost of software quality [15], but it looked only at *Java* projects hosted on *SourceForge.net*, limiting its generalizability to projects in other programming languages, or platforms like GitHub. Although Borges *et al.* considered new feature releases as one of the variables that could influence repository popularity, as part of its study on popularity versus key repository characteristics and traits of a software project [1], it neglected to take the code coverage levels within each feature release into consideration.

Understanding the relationship between code coverage and popularity is crucial for advancing software development practices, as increases in code coverage have reduced the number of defects and bugs that are detected, leading to higher quality code [12]. If positive relationships exist between code coverage and popularity, software projects that have not yet adopted or used coverage testing tools regularly to increase their code coverage could be more effectively convinced to do so, due to the benefits of increased popularity in the long run. In addition, the end user (the consumer) can also benefit from high code coverage levels, through better quality and reliable software products that many rely on in their daily lives.

This intriguing gap compels us to analyze how improvements in code quality, as measured by code coverage of branches, statements, functions, and lines [16] using automated coverage testing tools like *Codecov* and *Coverall*, are associated with changes in *stars*, a key metric [1] [17] for measuring project popularity on GitHub. By focusing on the code coverage metrics within popular repositories, new insights could be potentially unearthed about the factors that can contribute to the success and recognition of software projects. Our research could assist developers in finding new ways to improve their projects' quality and appeal, as we build upon prior project popularity studies that focused on other aspects of software quality. By clarifying the link between project popularity and code coverage, we aim to encourage developers to adopt better software development practices in

their pipelines, like code coverage testing.

We hypothesize that increased popularity may lead to enhanced code coverage, as another part of the existing broader commitment to quality within highly-rated projects [18]. Our theory posits that increased code coverage percentages within a project's codebase signify higher quality code, potentially fostering greater popularity for the project repository in terms of stars. This premise is grounded in the notion that superior code quality, attributed to a rigorous testing regimen, is a crucial factor in attracting and retaining quality software developers to the repository [18].

Our goal is to explore the relationship between code coverage and project popularity on GITHUB, aiming to unveil insights that can assist developers in augmenting the visibility and influence of their projects, potentially contributing to breakthroughs in terms of both the quality and attractiveness of software development as a whole.

To achieve this goal, we will address the following research questions:

- RQ 1: As an individual project becomes popular, what are the trends of code coverage percentages within the codebase of that project? For instance, as projects become more or less popular over time, does code coverage increase, decrease, or stay the same?
- RQ 2: Do stable and higher code coverage percentages in a project's source codebase correlate with higher project popularity on GITHUB?
- RQ 3: What role does introducing new features with code coverage play in shaping the popularity trajectory of software projects on GITHUB?

In the rest of the paper, we address related work, explain our methodologies, discuss our results, go over the threats to validity, and look ahead with our future work plans. Our code base can be found at **https://github.com/rajkunamaneni/Code-Coverage-Trends**.

II. RELATED WORK

Some researchers have analyzed key metrics of project repositories to understand and predict the popularity of projects [1], [17], [19]. Other studies examined the influence of standard folders [20] and company involvement [15] on software quality and popularity. Machine Learning has also been used to understand the popularity of GITHUB repositories from a predictive point of view, such as via the use of Multiple Linear Regressions [17] or Recurrent Neural Networks [21], both of which focused their research direction on the accuracy of their respective models in predicting popularity.

Our study examines the link between code coverage metrics and project popularity, bridging a research gap. We aim to provide insights for developers to enhance project visibility and quality in the open-source community, complementing existing literature on project success.

A. Trends of Code Coverage in Popular Projects

The investigation into key metrics of project repositories has been pivotal in understanding the importance of project popularity [19], finding that the number of branches, open issues, and contributors had a significant effect size with regards to the identification of popular projects. Another study has found that statistically significant correlations exist between a project's appeal and its fork count, contributor engagement, and commit frequency, suggesting that these elements are indeed relevant indicators of a project's traction within the open-source community [1]. They discovered that the popularity of a repository strongly correlated with the number of forks, as well as the number of contributors and commits to a weaker extent. Through its correlational findings, which encompassed metrics of a project repository - *age*, *commits*, *contributors*, and *stars*, the study was able to conclude that a large contributor userbase was essential for the success of an open-source project, thereby underscoring the necessity of an active and robust contributor base for thriving open-source initiatives.

Prior work has shown that language-specific characteristics like design pattern usage are not guaranteed to have the best indicators of code quality [10], leaving code coverage metrics one of the few key measures of code health and quality [11], [12], [13]. Code quality is an equally critical concern on the business side of things. Lower quality of code has led to more defects in the codebase, longer Time-in-Development, and increased unpredictability in terms of task completion, leading to negative impacts in terms of both time to market as well as external product quality in the form of unresolved bugs and software defects [6]. Yet, the relationship between a project's popularity surge and its code coverage has remained relatively uncharted. This gap presents a unique avenue for inquiry, proposing a hypothesis that an uptick in popularity could be accompanied by enhanced code coverage practices, reflecting a project's amplified commitment to maintaining or improving code quality amidst growing visibility and scrutiny.

The findings from Capra *et al.* provide additional valuable context for our research. While the authors explored the trade-offs between popularity and structural quality in cooperate-driven open source projects [15], we extend from their existing findings by exploring how popularity in a community-driven context might influence one key aspect of code quality, code coverage, over time. The notion that cooperate-driven projects might prioritize quality when compared to other projects, was defeated in their findings, as the structural quality of software degraded due to firms imposing "corporate constraints [on open source] developing practices" [15], although they reveal an important silver lining: "firms are significantly investing in OS projects and that they may play a crucial role for projects' popularity" [15]. This silver lining provides a foundational perspective for addressing RQ 1. If community-driven projects, which could have different value systems (compared to profit-focused cooperate projects) that are potentially reflected in their code coverage values, then increases in popularity might drive improvements to their code coverage and thus project quality as a whole. By examining the trends of code coverage within popular GITHUB projects across their life cycle, we can assess whether they emphasize structural quality by having consistent, robust code coverage practices. Our research will extend beyond company involvement in projects by looking into the connection between popularity and code coverage in

both community-based and company-driven software repositories that are hosted on GITHUB to gain a complete picture of the open-source software engineering landscape.

In a study on project popularity and code coverage, understanding how the size of the developer community, project activity levels, and user engagement affect code coverage can shed light on the significance of community-driven factors in ensuring comprehensive test coverage and code quality [22]. Ray *et al.* suggests that the development process, including practices around code testing and coverage, might be more indicative of code quality trends in popular projects than the choice of programming language [22]. This perspective further encourages us to seek a detailed examination of development practices in popular repositories to understand how code coverage trends evolve. Their research highlights that the modest effects of language design on software quality are overshadowed by process factors, suggesting a need for further exploration into the interplay between repository popularity and activity, and code quality. This is an important insight that our analysis of code coverage across GITHUB repositories can aim to address, by looking at changes in code coverage when taking the repositories' number of stars and lines of code added or deleted for each commit into account.

B. Correlation Between Code Coverage and Project Popularity

The exploration into structural and organizational facets of repositories, including the role of standard folder structure [20] and the impact of corporate sponsorship or involvement [15], has shed light on their potential effects on software quality and, by extension, possibly even popularity. However, the discourse around the interplay between code coverage percentages and project popularity on platforms like GITHUB remains nonexistent. It stands to reason that projects marked by higher and more consistent code coverage rates might also command greater popularity, attributed to the perceived or tangible elevation in quality that they offer, as we discussed in Section II-A. Before our research, it remained an open question if having a high or stable code coverage percentage attracted more people to a project, thus making it more popular. Exploring this question could help us understand why some open-source projects become popular. In addition, it might also show that a project's code coverage level is an important factor in determining how attractive and valuable people see these projects.

The positive correlation between documentation efforts and popularity [7], as identified by Aggarwal *et al.*, enriches our exploration for RQ 2. It suggests that just as high-quality documentation can elevate a project's popularity, so too might high code coverage, as both are indicators of a project's commitment to quality. This leads us to scrutinize whether projects with high code coverage are more likely to be popular on GITHUB, indicating that quality, as reflected in thorough testing practices, is a significant factor in attracting and retaining users and contributors.

The trade-offs highlighted by Capra *et al.* between software structural quality and popularity [15] resonates with RQ 2, which probes the potential correlation between stable or high code coverage percentages and project popularity. By extending the inquiry to code coverage, we seek to understand whether a robust testing regimen, as evidenced by high code coverage, could be a factor in a project's popularity on GITHUB, akin to the structural quality discussed by Capra *et al.*

By delving deeper into the relationship between code coverage and project attraction, we can refine strategies for project management and community engagement. Such insights could inform decision-making processes for developers and users alike, guiding resource allocation and investment in projects with the greatest potential for long-term success and impact.

C. Impact of New Features on Popularity and Code Coverage

The dynamics surrounding the introduction of new features within a project and their impact on its popularity have been acknowledged, with some efforts recognizing new feature roll-outs as a variable influencing project appeal [1]. Yet, these analyses have predominantly avoided the examination of the code coverage levels of these newly introduced features, leaving a void in comprehending how the qualitative aspect of new features — gauged through their code coverage — shapes the project's popularity trajectory.

Understanding the correlation between the introduction of new features, code coverage metrics, and the project's popularity can provide valuable guidance for developers and project managers alike. By examining how changes in code coverage influence the attractiveness and engagement levels of a project, we can refine strategies for feature development and deployment. This deeper analysis can also shed light on the importance of maintaining high code coverage levels throughout the development process to sustain and enhance a project's visibility and appeal over time. Such insights would not only benefit individual projects but also contribute to the broader discourse on effective practices in open-source software development.

This research void beckons an exploration into how the introduction of new features, as distinguished by their code coverage metrics, molds the attractiveness and engagement level of software projects on platforms like GITHUB. Delving into this aspect could unveil insights into development practices that not only augment a project's functionality but also bolster its overall appeal and standing in the open-source domain, thereby contributing to a more holistic understanding of factors that fuel a project's success and visibility.

For RQ 3, the insights from the study by Ray *et al.* into the modest effects of language design on software quality [22], contrasted with the significant impact of process factors, thereby guiding our investigation into the effects of new feature introductions on project popularity. We are prompted to consider how the process of integrating and testing new features — just like how it was done in the paper, but now reflected in terms of code coverage metrics — influences the attractiveness and user engagement of software projects.

In our research, we aim to fill the gaps left by the existing studies, by exploring how code quality, as measured by code

coverage, relates to the attractiveness and success of open-source projects. We'll analyze how code coverage evolves as projects become more popular, study the connection between code coverage and project appeal, as well as examine how adding new features with different code coverage levels affects project popularity. Through this work, we hope to provide valuable insights for developers to improve both the visibility and quality of their projects in the open-source community.

III. Methodology

In this section, we will describe the methods used to collect statistical data from popular repositories on GitHub, as well as our pipeline[2] for processing and analyzing data. Table I states our Null and Alternative Hypotheses for each Research Question.

A. Collecting the Repositories

To understand whether code coverage influences the popularity of the repository at hand, we first identified the set of programming languages that fit our domain of interest. Specifically, we excluded the languages that are not considered to be used as general-purpose languages, such as `CSS`, `HTML`, `Shell`, `Tex`, and `Powershell`. Instead, we focus on the mainstream programming languages that support the use of code coverage frameworks, such as `JavaScript`, `TypeScript`, `Java`, `Python`, and others.

Once we determined which programming languages would be a part of our domain of study, we then obtained the top 1000 most popular repositories based on the number of stars for *each* programming language within our domain, since stars are shown to be an acceptable metric for measuring the popularity of a repository [1] [17]. In addition, getting the top 1000 repositories for each language will provide us with more headroom for filtering and refining the repositories later on in the pipeline. The process of scraping for repositories based on the number of stars is done using a modified version of an existing GitHub repository scraper tool, which we adopted from *GitHub Ranking*[3], which employs the GitHub's GraphQL API to directly obtain repository data from their servers.

The list of repositories contains the key metrics about each of the repositories such as the repository name, the number of stars, forks, and issues, the main programming language used, the URL, the GitHub account username of the repository owner, as well as the last commit timestamp and date. This entire list is populated into a CSV file, where each row is a GitHub repository. We will use this CSV file in the next step of our pipeline, especially with interacting with the code coverage tools' API.

B. Refining the Repositories and Collecting Data

We focus on the repositories that use automatic code coverage tools such as *Coverall* or *Codecov*, each of which has a robust API implementation for extracting code coverage information from the commits of a repository. By leveraging existing code coverage reports and data from code coverage APIs, we can avoid manually rerunning the code coverage commands for every repository, as well as the computational overhead that can result from executing the testers and re-generating coverage reports locally. Thus, we filter out any repositories that do not utilize automatic code coverage tools to perform code coverage, as well as those that don't perform code coverage analysis at all. Additionally, we excluded repositories that returned errors in their code coverage reports, most likely due to improper implementation of the code coverage tools on their end. Such errors include null or empty values within their JSON fields or returning exactly zero for coverage percentage. We also found that code coverage tools were not guaranteed to be executed for all commits within a repository that elected to utilize such tools, and sometimes only for commits within a specific time range. Therefore, we further limit our set of repositories for data analysis to those that have successfully generated code coverage data for the commits that span for at least three years. This ensures that we have sufficiently spanned temporal data to conduct robust time series analysis. This process is further outlined in Figure 1.

From our refined list of repositories (which contains 1804 GitHub repositories) and given the repository name and the username of the repository owner, code coverage percentages for each commit SHA hash that has been registered in the code coverage tools are returned. This list of commits (grouped by repository name, ordered by commit timestamp, each commit containing coverage percentage, timestamp, hash value, parent repository name, and username, as well as the number of stars at the time of the commit) is then populated into another CSV file which we use for the next phase of our pipeline, data analysis.

Using the GitHub REST API with our refined set of repositories that used code coverage tools, we also collected Pull Request data for each repository, where each Pull Request entry consists of the submitter's username, Pull Request URL, issue URL, state, creation date, updated date, merge date, and the commit SHA (if applicable). Since the timestamp consisted of time and date, the time portion was stripped out since we are focusing on the day-to-day PR history data. Each of the repository's PR data is stored in another CSV file that is specific to that repository. The data within that CSV file is aggregated and converted into the number of pull requests per day, which is then appended and aligned with each of the unique commit dates for every repository that exists in the master report CSV file. This makes up a new column with the number of pull requests per day, allowing us to perform a time series analysis with another variable and see if correlations exist between pull requests and code coverage percentages, as part of our multivariate data analysis.

In a similar manner to the historical Pull Request data, we also appended the data for the number of contributors per day for each repository to the master report CSV file. Thus, we now have the following fields in that file: *username, repository name, code coverage percentages, hash of the commit on that date, timestamp date, language, daily star, total star, additions*, and *deletions*, as well as the last two fields that we appended,

[2]https://github.com/rajkunamaneni/Code-Coverage-Trends/tree/main/src
[3]https://github.com/EvanLi/Github-Ranking

RQ	Metric	Null Hypothesis	Alternative Hypothesis
RQ1	Code Coverage and Popularity	There is minimal correlation between code coverage percentage and repository popularity	An increase in repository popularity correlates strongly with an increase in code coverage percentage on GITHUB
RQ2	High Code Coverage and High Popularity	There is a negligible correlation between stable and higher code coverage percentages in a project's source codebase and its popularity on GITHUB	There is indeed a correlation between stable and higher code coverage percentages in a project's source codebase and its popularity on GITHUB
RQ3	New features with Code Coverage and Popularity	The introduction of new features with high or low code coverage does not significantly affect the popularity trajectory of software projects on GITHUB	The introduction of new features with high or low code coverage does significantly affect the popularity trajectory of software projects on GITHUB

Table I: Hypothesis Testing Methods for Code Coverage and Popularity on GITHUB

Number of pull requests on Timestamp and the *Number of Contributors on Timestamp*. With this comprehensive commit data that is organized in a centralized file, we proceeded to construct individual datasets for each repository, organized by date. For every date, we tabulated the count of distinct users who committed to the repository, which we designated as our contributor count for the corresponding dataset. Subsequently, we saved each dataset as a CSV file specific to its respective repository.

C. Popularity

Inspired by the work of Borges *et al.* [1], who utilized the number of stars (indicating user satisfaction and interest) of GITHUB repositories as a metric for measuring software popularity, we adopted a similar approach in our paper to gauge the popularity of software systems. To assess popularity, we gathered time series data on the number of stars for these 1804 GITHUB repositories with code coverage information, selecting samples corresponding to instances when the tools provided coverage reports. Through this examination, we aim to uncover patterns and insights that address RQ 1, 2, and 3.

Building upon the methodology as set forth by Borges *et al.*, we encountered and overcame the 40k star restriction imposed by the GITHUB REST API. To surpass this limitation, we utilized Graph QL, enabling us to retrieve information for repositories exceeding the 40k threshold without constantly interacting with the traditional GITHUB REST API.

D. Statistical Methods

To address each of our RQs, we employed Uniform Distribution Sampling from the *NumPy python* library to ensure an even distribution of data points throughout the project's duration. This approach is crucial for accurately capturing the evolution of code coverage as new features are introduced. By evenly sampling data points, we can effectively monitor changes in code coverage over time and assess the impact of feature introductions on the project. Ensuring a well-spread distribution of data points is essential as it prevents the clustering of observations from consecutive days. This avoids potential biases from arising from uneven sampling intervals and also provides a more representative picture of the project's development over time.

E. Dataset

Our dataset consists of code coverage results and time series data from 1804 repositories, categorized based on their star counts. Repositories with 10,000 stars or more were classified as High Popularity, those with star counts ranging from 1,000 to 10,000 were categorized as Medium Popularity, and repositories with fewer than 1,000 stars were considered Low Popularity. Specifically, there were 554 High Star Count Repositories, 1131 Medium Star Count Repositories, and 119 Low Star Count Repositories. The Popularity boxes in Figure 1 outline this approach. We filtered the data to exclude code coverage reports from repositories where the tool was tested but not extensively used. We also limited the scope to the last three years to focus on recent project data, although some repositories have data spanning since the inception of the project, allowing for long-term analysis.

F. Analytical Approach

To delve deeper into the relationship between code coverage and popularity, we followed a methodology akin to that of Aggarwal *et al.* [7], aligning the time series of popularity with code coverage reports. Our primary objective was to discern whether fluctuations in code coverage corresponded with shifts in popularity.

Addressing Research Question 1, we commenced with a linear regression analysis and utilized Pearson Correlation Analysis to investigate the connection between code coverage and popularity. Repositories were stratified into three clusters based on their popularity levels (*high*, *medium*, and *low*), enabling us to scrutinize trends in star growth alongside variations in code coverage over time. This clustering methodology allowed for the identification of overarching patterns and trends within our dataset.

By employing Pearson Correlation Analysis to examine the relationship between these variables, we can uncover patterns of changes and fluctuations. This analytical approach, reminiscent of the methodology used by Ray *et al.* [22], facilitated the identification of both positive and negative correlations between variables.

Research Question 1 explored the relationship between project popularity and code coverage across various repositories, providing a broad overview of how these factors correlate

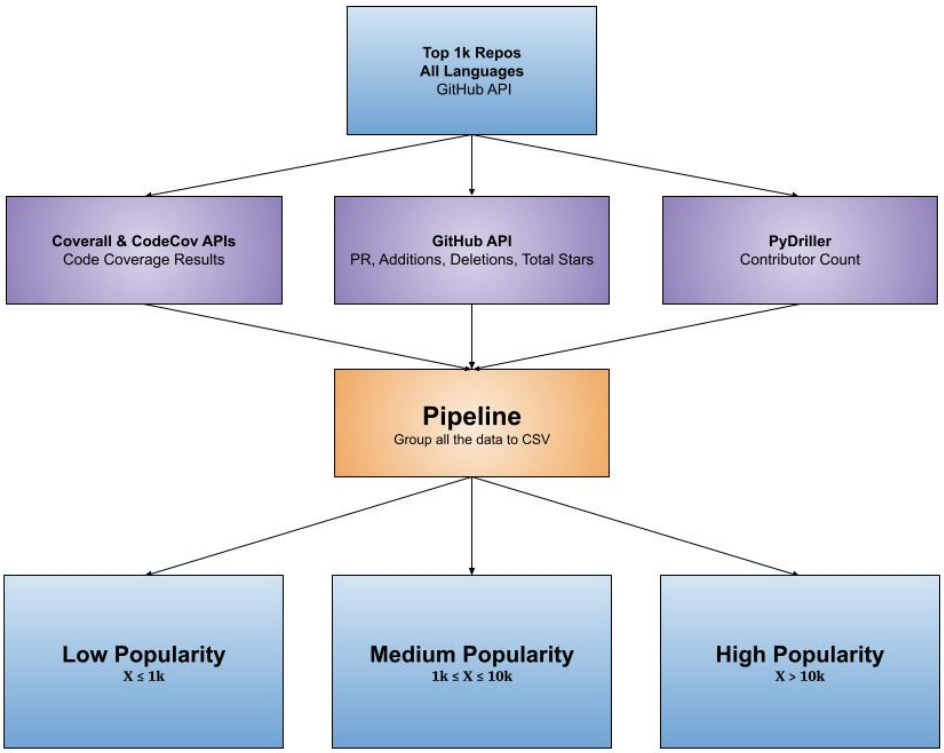

Figure 1: A visual overview of our data collection process with four major stages, along with the key variables considered. *X* is the Repository Popularity based on the *Total Star* count.

in different contexts. However, it did not directly address whether stable and higher code coverage percentages in a project's codebase correlate with higher project popularity on GITHUB, which is the focus of Research Question 2.

While RQ 1 laid the groundwork by examining correlations in a general landscape, RQ 2 delves deeper into understanding the specific relationship between code coverage and project popularity. It seeks to provide insights into whether projects with better code quality, as indicated by higher code coverage percentages, tend to garner more attention and adoption within the GITHUB community. Therefore, this RQ aims to provide a more targeted analysis to elucidate this aspect of software development dynamics.

We again conducted Pearson Correlation Analysis in order to specifically analyze this relationship, and also supplemented our findings with Root Mean Square Error (*RMSE*) to strengthen the validity of our results for addressing RQ 2. This will allow us to discern the extent to which code coverage impacts popularity among highly active repositories, shedding light on the dynamics that are at play between these variables.

For Research Question 3, we examine the relationship between new features and their impact on code coverage and repository growth. To understand the occurrence of feature additions and deletions, we rely on the notion that pull requests provide the most comprehensive insight, as established by Gousios *et al.* [23]. Since pull requests serve as a reliable indicator of feature additions [23], we analyzed the total number of pull requests within our dataset in a time series fashion to observe changes in code coverage. This approach allows us to understand how the evolution of code coverage, particularly in response to new feature introductions, impacts the overall popularity trajectory of software projects on GITHUB.

We conducted a Linear Regression analysis in conjunction with Pearson Correlation Analysis to assess whether the addition of new features affects code coverage and subsequently, the popularity of the project. This investigation aims to uncover whether the introduction of new features influences code coverage and, by extension, the popularity of the repository.

IV. RESULTS

A. RQ 1

To address Research Question 1, we examine the results obtained from Linear Regression (that was conducted on various repositories) to investigate the relationship between project popularity and code coverage. Table II provides detailed metrics for repositories categorized by *high*, *medium*, and *low* popularity levels.

For repositories categorized as having high popularity, such as *AdGuardHome* and *CocoaLumberjack*, we observe strong positive correlations between project popularity and code coverage. In the case of *AdGuardHome*, the correlation coefficient (0.98) indicates a highly significant positive relationship, suggesting that as the project gains popularity, code coverage tends to increase substantially. Similarly, *CocoaLumberjack*

Analysis Level	Analysis Methods for Investigating Code Coverage Versus Popularity on GitHub					
Level	**Repository**	**Coefficient**	**P-value**	**R Squared**	**Correlation Coefficient**	**RMSE**
High						
	AdGuardHome	0.0009111	4.596e-33	0.9509	0.9751	0.8246
	Charts	0.00357	2.546e-24	0.8865	0.9415	3.4634
	CocoaLumberjack	0.009434	2.702e-13	0.6748	0.8215	3.8801
	HikariCP	-0.0003753	1.139e-11	0.6206	-0.7878	1.1784
	NodeBB	0.0009943	1.875e-05	0.1167	0.3417	6.4914
	OCRmyPDF	0.0007424	0.003254	0.0570	0.2388	7.332
	Polly	0.007367	6.446e-11	0.5925	0.7698	2.5927
	faker	0.0007124	0.000311	0.2394	0.4893	5.8317
	falcor	0.0002541	0.2601	0.0263	0.1623	5.3306
Medium						
	BlueSocket	0.0603	1.50e-06	0.7527	0.8676	1.347
	CameraView	0.00894	2.32e-10	0.5705	0.7553	8.412
	Cataclysm-DDA	0.00475	3.89e-38	0.9698	0.9848	0.936
	CombineCocoa	0.00740	0.00195	0.3468	0.5889	1.361
	CopyQ	0.00269	7.35e-10	0.5497	0.7414	4.502
	EternalTerminal	0.00301	1.00e-13	0.6878	0.8294	0.644
	Flask-SocketIO	0.00423	2.17e-14	0.7070	0.8408	1.136
	ImageProcessor	0.00249	3.08e-05	0.4935	0.7025	0.384
	JJException	-0.01139	7.17e-09	0.6102	-0.7812	1.601
Low						
	IntroViews-Flutter	0.0192	8.41e-05	0.4966	0.7047	1.557
	abao	-0.021	1.30e-08	0.4935	-0.7025	1.579
	chopper	0.0129	3.05e-24	0.8856	0.9411	0.549
	dart-code-metrics	0.00764	1.04e-08	0.4981	0.7058	1.296
	easy_localization	-0.0607	3.49e-06	0.3643	-0.6036	4.715
	equatable	-0.00213	0.0617	0.0771	-0.2777	0.597
	floor	0.0172	2.58e-15	0.7317	0.8554	2.245
	flutter_typeahead	0.0375	0.00025	0.8293	0.9106	0.086

Table II: Investigating relationships between Code Coverage and Popularity on GitHub via Pearson Correlation (RQ 1)

exhibits a notable positive correlation (0.82), indicating a robust association between project popularity and code coverage improvement.

Conversely, repositories like *HikariCP* display negative correlations between popularity and code coverage. Despite having consistent popularity levels, *HikariCP* shows a significant negative correlation coefficient (-0.79), suggesting that as the project popularity remains stable, code coverage tends to decrease.

For repositories categorized as having medium popularity levels, a diverse range of correlations between project popularity and code coverage is observed. For instance, *CameraView* and *Cataclysm-DDA* exhibit positive correlations (0.76 and 0.98, respectively), indicating that an increase in project popularity tends to coincide with improved code coverage for these two repositories. Conversely, *JJException* displays a negative correlation coefficient (-0.78), suggesting that, despite moderate popularity, code coverage may decrease over time for this project. Repositories such as *Flask-SocketIO* and *CopyQ*, also demonstrate positive correlations between popularity and code coverage, albeit to a lesser extent (0.84 and 0.74, respectively).

For repositories categorized as having low popularity levels, a similar diversity in correlations between popularity and code coverage is observed. For example, *IntroViews-Flutter* and *flutter_typeahead* show positive correlations (0.70 and 0.91, respectively), suggesting that even with low popularity, code coverage tends to improve as project visibility increases. Conversely, repositories like *easy_localization* exhibit negative correlations (-0.60), indicating that code coverage may decrease despite low project popularity. Other low popularity repositories such as *floor* and *chopper*, display significant positive correlation coefficients with code coverage (0.86 and 0.94, respectively) as well.

We provide insight into the distribution of correlations within each popularity category. In the high popularity category, out of 350 repositories, 75% exhibit positive correlations, while 15% show negative correlations. 10% of the repositories in this category were excluded from the analysis due to them encountering issues with code coverage testing tools, such as misuse or initial trials, on a piloting basis, without extensive and regular adoption of such tools.

In the medium popularity category, out of 1044 repositories, 80% demonstrate positive correlations, while 10% exhibit negative correlations. Similarly, in the low popularity category, out of 103 repositories, 60% display positive correlations, while 20% exhibit negative correlations. As in the case of the high popularity category, 20% of the repositories in the low popularity category and 10% in the medium popularity category were also excluded from our analysis, due to the aforementioned issues that they likely encountered when trying to use the code coverage tools.

B. RQ 2

The correlation coefficients observed in the analysis provide quantitative measures of the relationship between code coverage percentages and project popularity on GITHUB. For repositories with positive correlation coefficients, such as *BayesianOptimization* (0.37) and *NodeBB* (0.34), there is a tendency for higher code coverage percentages to correspond with higher project popularity. Conversely, repositories with negative correlation coefficients, such as *equatable* (-0.28), suggest an inverse relationship between code coverage percentages and project popularity. Higher project popularity appears to coincide with lower code coverage percentages, indicating that factors other than test coverage might be influencing project visibility and adoption in this repository. Correlation coefficients close to zero, as seen in *SwifterSwift* (-0.06), indicate a weak linear relationship between code coverage and project popularity.

The data supporting these observations can be found in Table III. Out of the 200 repositories identified with both high popularity and high code coverage, 60% exhibited a weak linear relationship between these two metrics, while the remaining 40% showed a moderate linear correlation. This highlights the varied nature of the relationship between project popularity and code coverage, indicating that factors beyond code quality metrics alone may influence a project's visibility and adoption on GITHUB.

Repository	Correlation Coefficient	P-value	RMSE
ArduinoJson	0.0618	0.6697	1.0671
BayesianOptimization	0.3729	0.0076	4.0642
ChaiScript	-0.1729	0.2299	2.1932
NodeBB	0.3417	1.88e-05	6.4914
SwifterSwift	-0.0677	0.6403	4.0899
equatable	-0.2777	0.0617	0.5968
ffigen	0.1690	0.2408	6.4850
jiffy	0.3651	0.0174	2.3811

Table III: Analysis of the relationships between High Code Coverage and Growth in Popularity

C. RQ 3

In addressing Research Question 3, which examines the relationship between introducing new features with different code coverage levels and the popularity trajectory of software projects on GITHUB, we analyze the correlation coefficients presented in Table IV.

For example, repositories like *argo-workflows* and *air* show positive correlation coefficients of 0.85 and 0.87, respectively. This suggests that introducing new features with higher code coverage tends to positively impact the popularity trajectory of these projects. Conversely, repositories like *PaddleNLP*, with a correlation coefficient of -0.98, indicate a negative relationship between introducing new features with low code coverage and project popularity.

Repositories such as *TDengine* and *Charts* demonstrate strong positive correlations, with coefficients of 0.94 and 0.92, respectively. This underscores the importance of maintaining high code coverage levels when introducing new features, as it contributes to an enhanced popularity trajectory for software projects on GITHUB.

Similarly, repositories like *CocoaLumberjack* and *Catch2* exhibit positive correlation coefficients of 0.82 and 0.84, respectively. This suggests that the introduction of new features with higher code coverage values, positively influences project popularity over time.

It is also worth noting that out of the total 1496 repositories analyzed, approximately 72% show a positive correlation between the introduction of new features and code coverage, aligning with the observed pattern as outlined for most of the repositories in Table IV. However, around 28% of the repositories exhibit a negative correlation, indicating a decrease in code coverage during the introduction of new features based on pull requests.

Repository	Correlation Coefficient	P-value	RMSE
argo-workflows	0.8538	3.26e-15	0.5190
air	0.8691	2.76e-16	8.6329
TDengine	0.9446	7.09e-25	4.0724
Polly	0.7343	1.28e-09	2.7570
PaddleNLP	-0.9773	4.99e-34	0.4380
CocoaLumberjack	0.8223	2.41e-13	6.8642
Charts	0.9192	4.64e-21	4.0479
Catch2	0.8370	3.65e-14	2.3533

Table IV: Correlational relationships between Pull Requests with Code Coverage in terms of Growth in Popularity

V. DISCUSSION

A. RQ 1

The results from Table II show that for high popularity repositories such as *AdGuardHome*, *Charts*, *CocoaLumberjack* and *Polly*, an increase in the popularity of the project over time had a strong correlation with increased code coverage, with the rest showing lower correlational values, except for one repository (*HikariCP*), which had a negative correlation. For the medium popularity repositories, all repositories showed a strong positive correlation except for one - *JJException*. For the low popularity repositories, although we saw a higher number of repositories with negative correlations, such as *abao*, *easy_localization*, and *equatable*, there was still a sizable majority of repositories within this popularity category that had positive correlational values. This means that, since the vast majority of the repositories observed had a high correlation coefficient and a statistically significant p-value (with 82.17% having a positive value), we can therefore reject the null and accept the alternative hypothesis — that there exists a strong positive correlation between a project's popularity and its code coverage.

Upon deeper investigation into the repositories that had negative correlation coefficient values, there was a sudden drop in code coverage observed for these repositories (which could serve as one of the possible underlying causes of their negative correlation coefficients). This suggests that factors other than project popularity might be influencing code coverage trends within the repositories.

The most commonly observed trend seems to indicate that both code coverage and popularity increased over time, except

in a few cases where code coverage dropped over time. There are however a few notable exceptions where the null hypothesis can be accepted, such as in the case of *falcor* and *equatable*, where the p-value for the correlation was not statistically significant. This situation shows that changes in the values of popularity are not always correlated with changes in code coverage; that certain factors other than popularity may influence code coverage improvements as well.

From our results, we noticed that the repositories shared similar correlations and trends between project popularity and code coverage, regardless of whether they belonged to the *low*, *medium*, or *high* star count popularity groups, potentially indicating that the correlation between project popularity and code coverage might not be dependent on the popularity category. However, correlation is not to be mistaken with causation, as the correlation between these two variables (*popularity* and *code coverage*) does not indicate causation (*i.e.*, whether an increase in popularity causes an increase in code coverage or vice versa, cannot be proven with our results).

***RQ*1**: There is a significant correlation between project popularity and code coverage with the general trend being that increases in code coverage correlates with increases in popularity.

B. RQ 2

From Table III, the repositories are split almost evenly between weak and moderate correlations, with the rest indicating weak negative correlations, which means that high popularity and high code coverage do not have a very strong relationship with each other. This also suggests that, in addition to code coverage metrics, other factors may need to be taken into consideration as well. This could allow for a more complete understanding of the relationship between projects with better code quality and popularity, especially about other aspects that drive interest towards projects with high quality of code, beyond just code coverage. Thus, code coverage percentages may not strongly impact project popularity in these instances, or that other variables might play a more significant role.

For example, one of the relationships we investigated was the *NodeBB* repository, mentioned in Table III. Taking a deeper dive into the daily activity of that repository, it can be seen that there was a consistent increase in the number of pull requests and additions. When looking at the linear relationship between code coverage and popularity and the increase in other metrics like the additions and pull requests, it can be inferred that the code coverage should be taken into consideration along with other metrics discussed in [7], such as forks, pull requests and documentation. This, along with the almost even split between a weak linear relationship and moderate linear relationship, indicates that code coverage is not a strong variable to consider when investigating the possible correlations with repository popularity.

Another repository we investigated was the *ffigen* repository, also in Table III. Looking into this repository's daily activity, it can be seen that the total pull requests increased from 29 to 299 in the span of around 3 years from 2020 to 2023. Throughout the time investigated it can also be seen that there was a considerable amount of additions that were evenly spread apart. Again, this emphasizes that simply code coverage percentages alone might not be directly correlated with the level of popularity of the repository, rather, it should be coupled with other metrics — as previously mentioned — to form a more complete relationship between the various metrics of the repository and popularity. In this case, the number of additions or its significance thereof, could serve as possible indicators of a marked increase in interest and popularity of the project. Therefore, the findings suggest that we accept the null hypothesis, in that there is a negligible correlation between stable and higher code coverage percentages in a project's source codebase, and its popularity on GITHUB.

***RQ*2**: While there is a moderate association between high code coverage and popularity, relying solely on code coverage may not be sufficient in determining correlations with popularity, since other factors such as pull requests, forks, and features added, should also be taken into account.

C. RQ 3

The results from Table IV, demonstrate that most of the repositories have a positive correlational relationship between the number of pull requests and code coverage. In addition, having an increased number of pull requests seems to lead to higher code coverage percentages for a significant portion of the repositories that we studied. Based on our analysis of the pull requests from the repositories of Table IV, one explanation for this positive relationship could be attributed to the stringent and specific requirements that many of the projects imposed on individual developers.

As part of the repository's *continuous integration* (*CI*) pipeline, when individual developers submit pull requests in the first place, quality checks during the development process are enforced in an automated manner. This enforcement of quality and good practices increases the likelihood that only quality code with more complete code coverage will be accepted into the repository's codebase. Consequently, pull requests with low-quality code are more likely to fail the tests, leading to their eventual rejection or exclusion from the project's codebase, thereby disincentivizing pull requests with low code coverage from being created in the first place.

This further demonstrates the benefits of adopting *CI* in terms of the positive relationship between teams that effectively used *CI and* had higher numbers of created pull requests, and the more effective discovery of bugs. In fact, the analysis of our findings for RQ 3 conforms with a key existing study on the benefits of adopting *CI* with regards to software quality and developer productivity [4]. The study found that teams that adopted *CI* were not only more effective at merging pull requests but also had fewer rejected pull requests within their external contributor base. Most importantly, concerning code quality, development teams that adopted *CI* also found software bugs more effectively than those that did not [4], confirming our positive correlational trends between most of the popular repositories (that adopted effective *CI* pipelines with automated code coverage testing that their pull requests

had to undergo) which had higher code coverage levels and thus better code quality, and the popularity of that given repository.

Almost all of the repositories with a positive Correlation Coefficient enforced several key requirements in their PR acceptance process. In the case of *CocoaLumberjack*, the repository provides a "New Pull Request Checklist" that each Pull Request needs to follow. The checklist stipulates the required prerequisite quality checks, including running and passing the given test suite as well as running any new required tests to show that the feature or fix that the Pull Request author has written, works properly.

For the repositories *air*, *PaddleNLP*, and *Catch2*, each Pull Request requires the execution of *Codecov*. *Codecov* then automatically checks to see if all "*modified and coverable lines are covered by tests*" and also provides a link to the full coverage report, as part of their extensive *CI* pipeline. If the given Pull Request does not pass the required quality checks, including the code coverage ones, then GITHUB warns the reviewer with a message saying that "*required statuses must pass before merging*" since "*some checks were not successful*". Although some instances that had those warning messages might still get approved after extensive manual review in part by the reviewer with write access, many of these pull requests are nevertheless likely to encounter increased scrutiny. This means that the test cases essentially serve as quality checks, which will enforce the execution of some parts of the code in an automated manner within the pipeline. Thus, if the code coverage is excessively low or causes the existing coverage percentages to decrease significantly, then quality issues might have likely arisen.

On the other hand, *PaddleNLP* had a significant negative correlation coefficient, most likely due to the acceptance of many pull requests that did not pass all the given tests in their pipeline, especially the ones about code coverage. The repository's pull requests were likely being vetted by reviewers who did consider the level of code coverage to be an important deciding factor in their acceptance or denial. For instance, some of the pull requests that failed to meet the minimum *Codecov* thresholds for `codecov/project` and `codecov/patch` test case domains still obtained approval and got merged into the project codebase [4]. Almost all of the pull requests, regardless of their status, either failed to pass at least some of the tests within their *CI* pipeline [5], returned missing coverage warnings from *Codecov*'s coverage report [6], or would decrease the existing code coverage percentage [7], while other pull requests had different checks in their *CI* pipeline that did not even consider the level of code coverage as part of their approval [8]. This situation could potentially reduce incentives for developers to produce code that is well-tested in terms of code coverage levels.

Despite the decrease in code coverage, *PaddleNLP*'s popularity is still increasing, pointing to other factors that could be attributed to the popularity of certain repositories, beyond code coverage percentages. This also means that pull requests alone might not be best for looking at additions or deletions, as other variables, such as large commits within the project's core contributor base, can also modify the codebase with high numbers of additions and/or deletions.

Nevertheless, from our findings for the majority of the repositories, low-quality pull requests tend not to be created as much as high-quality ones, due to the aforementioned automated quality checks that require any new code contributions to be sufficiently tested (and thus more likely to be higher in terms of code coverage as well) if the Pull Request is to be accepted. Therefore, we can accept the alternative hypothesis and conclude that the introduction of new features with varying levels of code coverage in the form of pull requests, does indeed play a role in shaping the popularity of a repository on GITHUB.

> ***RQ3***: There is a significant positive correlation between the introduction of new features, in the form of the number of pull requests created, and the popularity of a repository.

D. Future Work

Several directions for future research can build upon the established correlation between code coverage and project popularity on GITHUB. One area of interest is gaining qualitative insights from developers using a potential survey-based study. This could provide context to the quantitative data, especially in instances where repositories showed negative correlations or were excluded due to improper tool usage. Understanding the perspectives and experiences of project maintainers and contributors, regarding code coverage practices and their perceived impact on project popularity, could enrich the current understanding of the dynamics at play.

We could also use controlled studies to look for possible causal relationships between popularity, code coverage, and pull requests. A controlled test can build upon our results as it retains the potential to find causality between correlated variables like popularity and code coverage over time. On the other hand, this might require the collection of more data pertaining to the additional variables — beyond the ones that we focused on in our study — to supplement and support the potential findings. Yet, this remains a significant future work initiative, as accounting for controlled and uncontrolled variables could help to determine if and where causality is present in the data.

Expanding the scope of the study to a broader or wider period could show how code coverage and project popularity change with shifts in development practices, community engagement, and technological advancements, from a more substantial, longer-term point of view. Such an approach might also uncover the life cycle of software projects and identify critical phases where code coverage has a significant impact on popularity. Furthermore, considering projects hosted on platforms other than GITHUB, such as GITLAB or BITBUCKET, could provide insights into whether the trends are platform-specific or have broader principles in open-source development. At the same time, extending these areas of

[4]https://github.com/PaddlePaddle/PaddleNLP/pull/7911
[5]https://github.com/PaddlePaddle/PaddleNLP/pull/7451
[6]https://github.com/PaddlePaddle/PaddleNLP/pull/7985
[7]https://github.com/PaddlePaddle/PaddleNLP/pull/6035
[8]https://github.com/PaddlePaddle/PaddleNLP/pull/7913

scope can address the issues of generalizability that several studies have pointed out in the past as well, especially in terms of the representativeness of the data [4], [6], version control platforms covered [15], as well as the domain [24]. Along with potential improvements in the generalizability of our current findings, this analysis could also reveal platform-agnostic strategies on a universal basis across diverse groups of developers, beyond just platform-specific features that influence project success and code quality in a uniquely particular manner.

The negative correlation between code coverage and popularity needs a more in-depth investigation as well. A detailed case study of these repositories could show specific factors, such as substantial changes to the codebase, shifts in the contributor base, or changes in project direction and value systems, that might help to explain the reasons behind the decrease in code coverage in the face of rising popularity within certain repositories. Exploring external factors that could influence a project's popularity, such as marketing initiatives, community events, or endorsements in the tech community, could provide a more comprehensive understanding of the factors that contribute to a project's success via increased popularity beyond code quality. This could be in the form of a potential future study centering on answering why certain repositories have decreasing code coverage, yet they still exhibit signs of increasing popularity, with an in-depth analysis of a wide variety of external, social, and business factors.

The challenges developers face in using code coverage tools within their development workflows are also worth investigating. Insights gained could lead to improvements in tool design and best practices for the integration of various software development processes. For instance, examining the relationship between contribution patterns, project governance models, and their impact on both code coverage and project popularity could reveal how different management and engagement strategies influence the outcomes of software projects. This could provide valuable guidance for project leaders aiming to enhance not just code quality but also community engagement as a whole. Future research can deepen the understanding of the interplay between technical aspects of software quality, such as code coverage, and the complex social dynamics that contribute to the popularity and success of open-source software projects on platforms like GITHUB and others, potentially bridging the gap between Software Engineering and other fields of study, such as Socioeconomics.

E. Conclusion and Threats to Validity

In this study, we analyzed a dataset comprising 1804 GITHUB repositories to examine the interplay between code coverage and project popularity, coupled with other important repository and software metrics. Our investigation revealed a robust correlation between increasing code coverage and rising popularity. However, a deeper examination of the association between high code coverage and popularity unveiled a nearly even distribution of moderate linear relationships, indicating that elevated code coverage alone may not guarantee increased popularity. This underscores the necessity of considering additional metrics when exploring the factors that foster the increase in the popularity of software projects.

Moreover, we delved into the impact of code coverage and pull requests on repository popularity. Our analysis uncovered a significant positive correlation between the number of pull requests with varying levels of code coverage, and repository popularity. These findings collectively suggest that while code coverage may contribute to the increased popularity of a repository, its influence is contingent upon the consideration of other essential repository metrics. Thus, our study underscores the importance of designing a holistic approach that integrates various data metrics to comprehensively analyze repository success.

In our analysis, we recognize several threats to the validity of our findings, which could affect the robustness and generalizability of our conclusions. These considerations are crucial when interpreting our results and drawing insights from our research.

Concerning the threat to construct validity, one potential limitation lies in our use of pull requests as a proxy for understanding the introduction of new features. While pull requests are commonly used for this purpose, it's important to note that they can encompass various types of changes beyond the addition of new features, such as "unimpactful" pull requests that do not affect the overall design or structural quality of the codebase in a significant manner [25]. These kinds of pull requests may include other changes such as bug fixes, refactoring, documentation updates, as well as minor or trivial modifications.

Another noteworthy concern, in terms of the threat to external validity, pertains to the quality and scope of the data used in our analysis. Despite our best efforts to ensure the accuracy and completeness of the data retrieved from GITHUB repositories using a large dataset of repositories for each programming language within our domain, as described in Sections III-A and III-E, there may very well still be instances of missing or inaccurate information. Our analysis also focused on a specific subset of repositories, which may not fully represent the sheer diversity of projects on GITHUB, let alone projects hosted on other version control platforms as mentioned in Section V-D.

Finally, it's essential to acknowledge the presence of external factors and confounding variables, in our threat to internal validity, that may influence repository popularity but were not explicitly addressed in our analysis. This is especially true in the field of Software Engineering, where a large number of possibly confounding factors can exist due to its knowledge-intensive nature [24]. In our case, these factors could include external marketing efforts, community engagement initiatives, changes in technology trends, and broader socio-economic factors.

VI. TEAM MEMBERSHIP AND ATTESTATION OF WORK

Team members Raj Kunamaneni, Henry Chou, Anant Vishwakarma, Georgy Zaets, and Srivatsan Srikanth participated sufficiently.

REFERENCES

1 Borges, H., Hora, A., and Valente, M. T., "Understanding the factors that impact the popularity of github repositories," in *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2016, pp. 334–344.

2 Yu, Y., Yin, G., Wang, H., and Wang, T., "Exploring the patterns of social behavior in github," in *Proceedings of the 1st International Workshop on Crowd-Based Software Development Methods and Technologies*, ser. CrowdSoft 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 31–36. [Online]. Available: https://doi.org/10.1145/2666539.2666571

3 Tsay, J., Dabbish, L., and Herbsleb, J., "Influence of social and technical factors for evaluating contribution in github," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 356–366. [Online]. Available: https://doi.org/10.1145/2568225.2568315

4 Vasilescu, B., Yu, Y., Wang, H., Devanbu, P., and Filkov, V., "Quality and productivity outcomes relating to continuous integration in github," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2015. New York, NY, USA: Association for Computing Machinery, 2015, p. 805–816. [Online]. Available: https://doi.org/10.1145/2786805.2786850

5 Lu, Y., Mao, X., Li, Z., Zhang, Y., Wang, T., and Yin, G., "Internal quality assurance for external contributions in github: An empirical investigation," *Journal of Software: Evolution and Process*, vol. 30, no. 4, p. e1918, 2018, e1918 smr.1918. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.1918

6 Tornhill, A. and Borg, M., "Code red: The business impact of code quality - a quantitative study of 39 proprietary production codebases." Pittsburgh, PA, USA: IEEE, 2022. [Online]. Available: https://ieeexplore.ieee.org/document/9804508

7 Aggarwal, K., Hindle, A., and Stroulia, E., "Co-evolution of project documentation and popularity within github," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 360–363. [Online]. Available: https://doi.org/10.1145/2597073.2597120

8 Zhang, C. and Budgen, D., "What do we know about the effectiveness of software design patterns?" *IEEE Transactions on Software Engineering*, vol. 38, no. 5, pp. 1213–1231, 2012.

9 Haythornwaite, C., "Gamma, e., helm, r., johnson, r. & vlissides, j. design patterns: Elements of reusable object oriented software. new york: Addison-wesley, 1995." 2002. [Online]. Available: https://api.semanticscholar.org/CorpusID:62414717

10 Khomh, F. and Gueheneuc, Y.-G., "Do design patterns impact software quality positively?" in *2008 12th European Conference on Software Maintenance and Reengineering*, 2008, pp. 274–278.

11 Horgan, J., London, S., and Lyu, M., "Achieving software quality with testing coverage measures," *Computer*, vol. 27, no. 9, pp. 60–69, 1994.

12 Williams, T., Mercer, M., Mucha, J., and Kapur, R., "Code coverage, what does it mean in terms of quality?" in *Annual Reliability and Maintainability Symposium. 2001 Proceedings. International Symposium on Product Quality and Integrity (Cat. No.01CH37179)*, 2001, pp. 420–424.

13 Malaiya, Y., Li, M., Bieman, J., and Karcich, R., "Software reliability growth with test coverage," *IEEE Transactions on Reliability*, vol. 51, no. 4, pp. 420–426, 2002.

14 Sajnani, H., Saini, V., Ossher, J., and Lopes, C. V., "Is popularity a measure of quality? an analysis of maven components," in *2014 IEEE International Conference on Software Maintenance and Evolution*, 2014, pp. 231–240.

15 Capra, E., Francalanci, C., Merlo, F., and Rossi-Lamastra, C., "Firms' involvement in open source projects: A trade-off between software structural quality and popularity," *Journal of Systems and Software*, vol. 84, no. 1, pp. 144–161, 2011, information Networking and Software Services. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016412121000244X

16 Ivankovic, M., Petrovic, G., Just, R., and Fraser, G., "Code coverage at google," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 955–963.

17 Borges, H., Hora, A., and Valente, M. T., "Predicting the popularity of github repositories," in *Proceedings of the The 12th International Conference on Predictive Models and Data Analytics in Software Engineering*, ser. PROMISE 2016. ACM, Sep. 2016. [Online]. Available: http://dx.doi.org/10.1145/2972958.2972966

18 Börstler, J., Störrle, H., Toll, D., van Assema, J., Duran, R., Hooshangi, S., Jeuring, J., Keuning, H., Kleiner, C., and MacKellar, B., ""i know it when i see it" perceptions of code quality: Iticse '17 working group report," in *Proceedings of the 2017 ITiCSE Conference on Working Group Reports*, ser. ITiCSE-WGR '17. New York, NY, USA: Association for Computing Machinery, 2018, p. 70–85. [Online]. Available: https://doi.org/10.1145/3174781.3174785

19 Han, J., Deng, S., Xia, X., Wang, D., and Yin, J., "Characterization and prediction of popular projects on github," in *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, vol. 1, 2019, pp. 21–26.

20 Zhu, J., Zhou, M., and Mockus, A., "Patterns of folder use and project popularity: A case study of github repositories," *International Symposium on Empirical Software Engineering and Measurement*, 09 2014.

21 Hajiakhoond Bidoki, N., Sukthankar, G., Keathley, H., and Garibay, I., "A cross-repository model for predicting popularity in github," in *2018 International Conference on Computational Science and Computational Intelligence (CSCI)*, 2018, pp. 1248–1253.

22 Ray, B., Posnett, D., Devanbu, P., and Filkov, V., "A large-scale study of programming languages and code quality in github," vol. 60, no. 10. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3126905

23 Gousios, G., Zaidman, A., Storey, M.-A., and van Deursen, A., "Work practices and challenges in pull-based development: The integrator's perspective." Florence, Italy: IEEE, 2015. [Online]. Available: https://ieeexplore.ieee.org/document/7194588

24 Zhu, J., Zhou, M., and Mockus, A., "Effectiveness of code contribution: from patch-based to pull-request-based tools," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2016. New York, NY, USA: Association for Computing Machinery, 2016, p. 871–882. [Online]. Available: https://doi.org/10.1145/2950290.2950364

25 Barbosa, C., Uchôa, A., Coutinho, D., Falcão, F., Brito, H., Amaral, G., Soares, V., Garcia, A., Fonseca, B., Ribeiro, M., and Sousa, L., "Revealing the social aspects of design decay: A retrospective study of pull requests," in *Proceedings of the XXXIV Brazilian Symposium on Software Engineering*, ser. SBES '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 364–373. [Online]. Available: https://doi.org/10.1145/3422392.3422443