Analyzing GitHub Repositories: Community Interaction Patterns and the Impact of Comment Types on Star Growth

Josh Moody, LING 581

Abstract

This paper investigates community interaction patterns within GitHub repositories, specifically examining how different types of comments impact the growth of software projects. Utilizing natural language processing techniques, the study analyzes code comments and issue comments from a diverse selection of GitHub projects to address key research questions regarding the nature of technical communication, its variation across projects, and its influence on project popularity. Data collected from comments were classified into various categories using machine learning models and sentiment analysis. The study found that technical communication is predominantly explanatory in nature, with significant variations in communication styles across different projects. Furthermore, a linear regression analysis reveals that certain types of comments, particularly explanatory and future work comments, positively affect the rate of project growth as measured by GitHub stars. This research contributes to the field of software repository mining by highlighting the subtle yet statistically significant influences that developer interactions have on software project success. The findings suggest that effective management of technical communication can potentially enhance project visibility and growth. (written by ChatGPT)

Introduction

GitHub is a massively popular online service that helps developers manage and collaborate on their code. As projects on GitHub grow more popular, the need for effective documentation and collaboration grows to ensure the software remains dependable and understandable.

Among the most prevalent tools for documentation and collaboration are code comments and issue comments. Code comments are snippets of natural language embedded into source code files to clarify the purpose of the underlying code. Issue comments are topical messages left by developers or users in “issue tracking” software (e.g., GitHub Issues) to report bugs, ask questions, or request features.

By leveraging natural language processing (NLP) techniques, valuable insights into the software development process can be extracted from these comments. This process is known as “software repository mining.” This paper seeks to expand the body of knowledge in this domain by finding answers to the following questions:

- What does technical communication look like?
- How does technical communication differ between projects?
- How does technical communication affect the growth of software projects?

Related Work

Software repository mining is an active area of research that has yielded many valuable insights. For example, B. Yang et al. identified a positive correlation between developer sentiment and bug fixing speed in GitHub repositories (Bo Yang et al. 87). Additionally, V. Sinha et al. found a strong positive correlation between negative sentiment and the number of files changed per git commit (V. Sinha 552). However, the impact of developer language on project popularity, a primary focus of this project, has not been thoroughly studied.

Methods

This research project involved two primary phases: 1) dataset creation and 2) data analysis.

Dataset Creation

Creating the dataset was an involved process that involved gathering data and enhancing it with additional features generated with NLP. The following steps describe the full dataset creation process.

Step 1: Project Selection

A total of 19 GitHub projects were selected for analysis. To ensure generalizability of this study's findings, the projects were chosen to ensure diversity in purpose, size, and programming language. There is a slight bias towards generative AI projects and web frameworks due to the interests of this paper's author.

Software Project Name	**Software Purpose**	**Main Programming Language**
Sitcom Simulator	AI video generation	Python
Nodejs.org	Website for Node.js	TypeScript
sled	Embedded database	Rust
Auto1111SDK	AI image generation	Python
Devika	AI software engineering	Python
System.Linq.Dynamic.Core	Querying data	C#
HTTPRequest	Networking	C++
Flask	Web framework	Python
React	Web framework	JavaScript
Turbo	Web framework	JavaScript
Rails	Web framework	Ruby
Vue.js	Web framework	TypeScript
MoviePy	Video editing	Python
Astro	Web framework	TypeScript
htmx	Web framework	JavaScript
Phoenix	Web framework	Elixir
Ethereum	Blockchain	Go
Bootstrap	Web framework	JavaScript
Django	Web framework	Python

Step 2: Raw Data Collection

Code and issue comments were extracted using the GitHub API and the GitPython library. Issue comments were collected using the GitHub API. Code comments were collected by cloning each repository and using regular expressions to extract code comments from every git commit in each project's history. This process is similar to the "Mimansa" repository mining framework proposed by Megha Mittal and Ashish Sureka (2014) except without taking "requirement" data sources (e.g., wikis, README files, etc.) into account.

Step 3: Comment Classification

Each comment was classified into categories using the BART transformer model. Since the purposes of code and issue comments differ, the possible categories differed between these comment types.

Comment Type	Category	Real Example
Code	Explanation	we can't run the select function on the first tab
Code	Deprecated	DEPRECATED - Do not use if you can avoid.
Code	Future work	TODO - support more request types POST, PUT, DELETE, etc.
Issue	Question	@tburrows13 this is the exact issue I am facing. Did you find any solution to this?
Issue	Conclusion	This is a duplicate of #919
Issue	Discussion	Sorry for the delay, I've approved but would like to give the chance for another reviewer to merge it.
Issue	Solution	You can leverage many of the latest models, paid and free through a single API at Openrouter.
Issue	Feature request	A low-hanging fruit and huge Feature boost would be adding Langsmith. Thanks!
Issue	Bug report	This bug still persists with 4-2-stable.

Additionally, sentiment analysis was performed on each comment, classifying each as either "POSITIVE" or "NEGATIVE" using the DistilBERT transformer model. BERT-style models perform well on GitHub issue classification tasks (Siddiq and Santos, 2022).

Due to the high processing power required for these models, a maximum of 1,000 code comments and 1,000 issue comments were randomly sampled from each repo to be classified.

Step 4: Historical GitHub Stars

Since GitHub's API does not provide long-term historical star counts, this data was collected via star-history.com instead. These values were linearly interpolated to approximate the star count at the exact moment of each comment's creation.

Data Analysis

To answer the research questions stated in this paper's introduction, the dataset was grouped into three-month periods. In each period, the proportions of different comment types were measured to gain a broad understanding of the makeup of technical communication and how it has changed over time. This analysis was performed both in aggregate and on a per-repository basis.

To understand the impact of comments on project growth, a linear regression model was used to estimate the effect of individual comments' sentiment and category on star growth rate three months later. Specifically, the logarithm of star growth was used due to its right-skewed nature.

The three-month period was chosen as a good middle ground between being able to see the long-term effects of a code comment while also being short-term enough for this study's findings to result in practical, actionable insights for developers.

Evaluation

The findings from the data analysis are described below with accompanying results, visualizations, and interpretations for each research question.

What does technical communication look like?

Each of the metrics gathered above was analyzed in aggregate and on a per-repo basis to determine what technical communication is composed of at a high level.

Comment N-Grams

The most common unigrams and bigrams were extracted from the dataset. The results reveal that code comments are much more technical in nature than issue comments. However, both are extremely likely to contain hyperlinks to GitHub.com.

The following tables list the most common n-grams in **code** comments.

Rank	Term	Count
1	use	512
2	object	472
3	function	471
4	value	460
5	set	440

Rank	Term	Count
1	make sure	90
2	github com	57
3	license bsd	56
4	copyright 2010	40
5	return value	39

Interestingly, the unigrams *object* and *function* are used almost equally. This suggests that neither nouns nor verbs take precedence in the minds of developers. It may also suggest that neither object-oriented nor functional programming is inherently more "natural" than the other.

The following tables list the most common n-grams in **issue** comments.

Rank	Term	Count
1	issue	1721
2	like	1095
3	think	1076
4	use	1013
5	just	1008

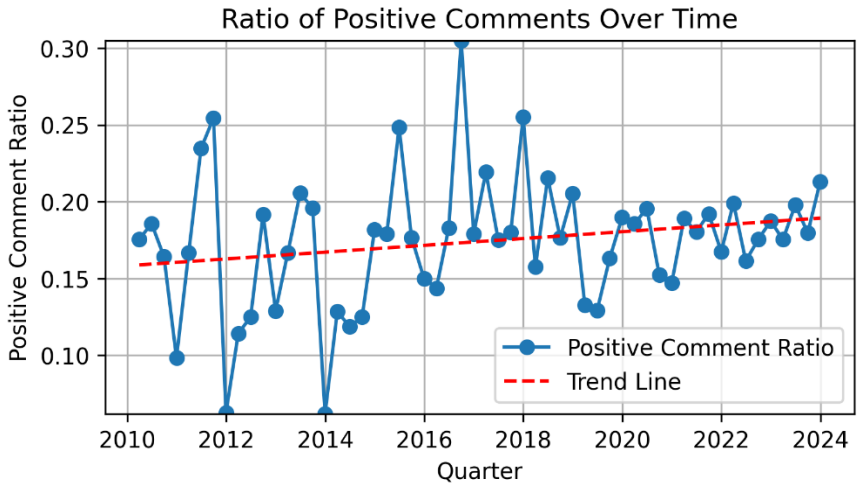
Rank	Term	Count
1	github com	487
2	https github	462
3	don think	145
4	looks like	133
5	use case	131

The unigram *issue* is almost twice as popular as the second most popular. This is likely because *issue* has two meanings in software development: it can refer to a generic problem with the software, or it can refer to a post on GitHub's issue tracking software.

A more complete list of common n-grams can be found in the appendix.

Comment Sentiment

As indicated in the following figure, most comments are negative in sentiment. Less than 20% of comments were classified as positive. However, positivity has been slowly increasing over the past decade, growing from 16% to 19%.



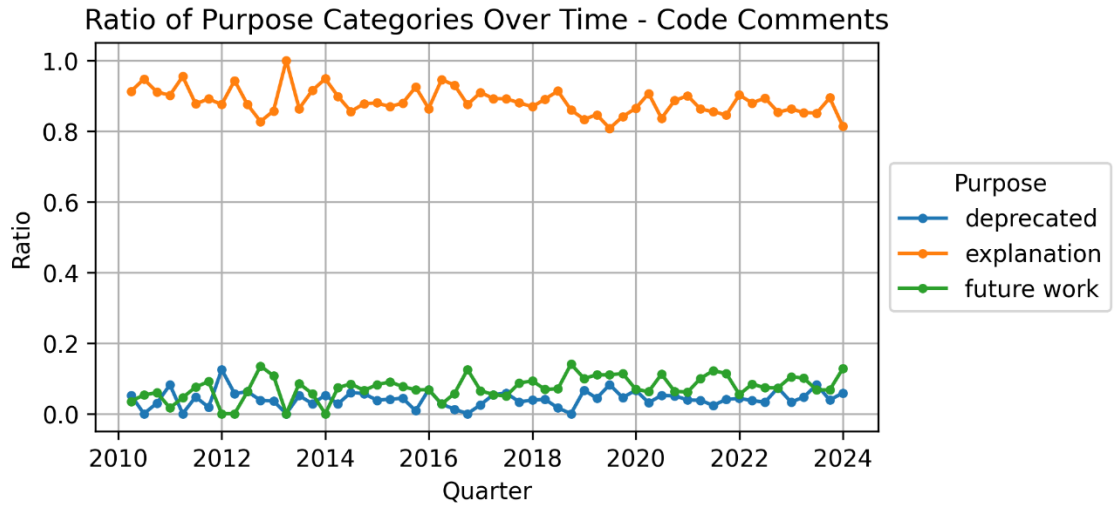
This negative slant may be because code errors and bugs are among the most frequently discussed topics. For example, the top twenty unigrams and bigrams contain "error", "problem", "breaking change", and "doesn work."

However, the strong negative slant is more likely a result of using an off-the-shelf sentiment analysis model for a specialized task. Prior research has found that general purpose sentiment analysis models perform poorly on software engineering tasks (Jongeling et al., 2017).

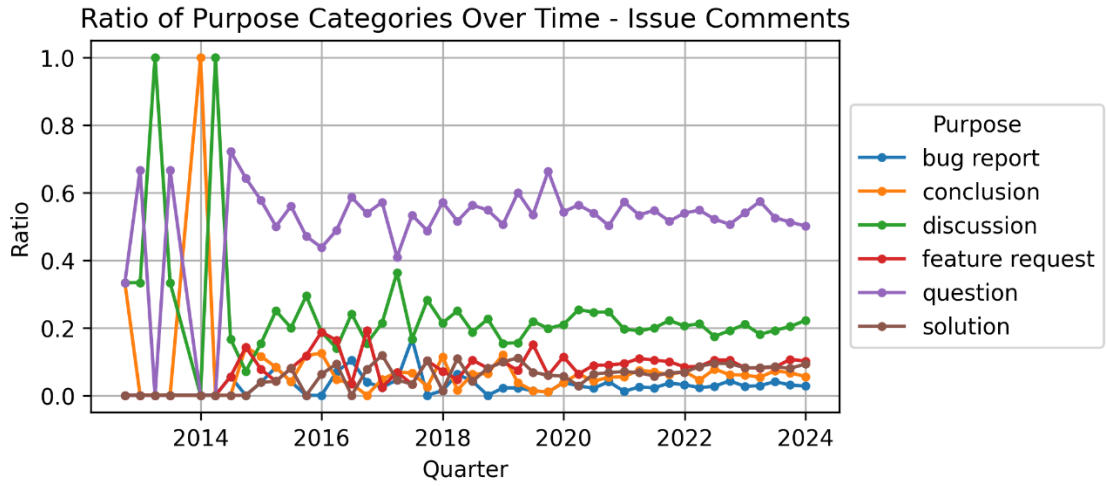
Comment Purpose

The relative frequency of each code comment type has remained mostly static over time, as shown below. Explanations are by far the most common purpose for code comments, accounting for over 80% of the total. Deprecation and future work comments are almost equal in frequency, but future

work takes a slight lead. This suggests that software projects tend to grow over time, with more features being created than destroyed.

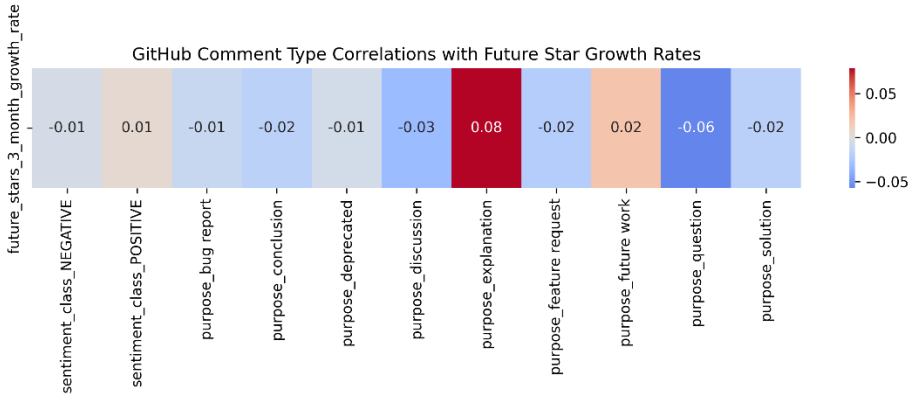


Similarly, issue comment purpose ratios have largely stayed the same over time. The instability of the values between 2013-2015 is due to the lower quantity of data available during those years and does not imply anything particularly interesting. More than half of issue comments are categorized as questions. This is interesting considering that only about 10% of comments are classified as solutions. Assuming the classification model is reasonably accurate, this indicates that many questions raised in GitHub issues go unanswered.

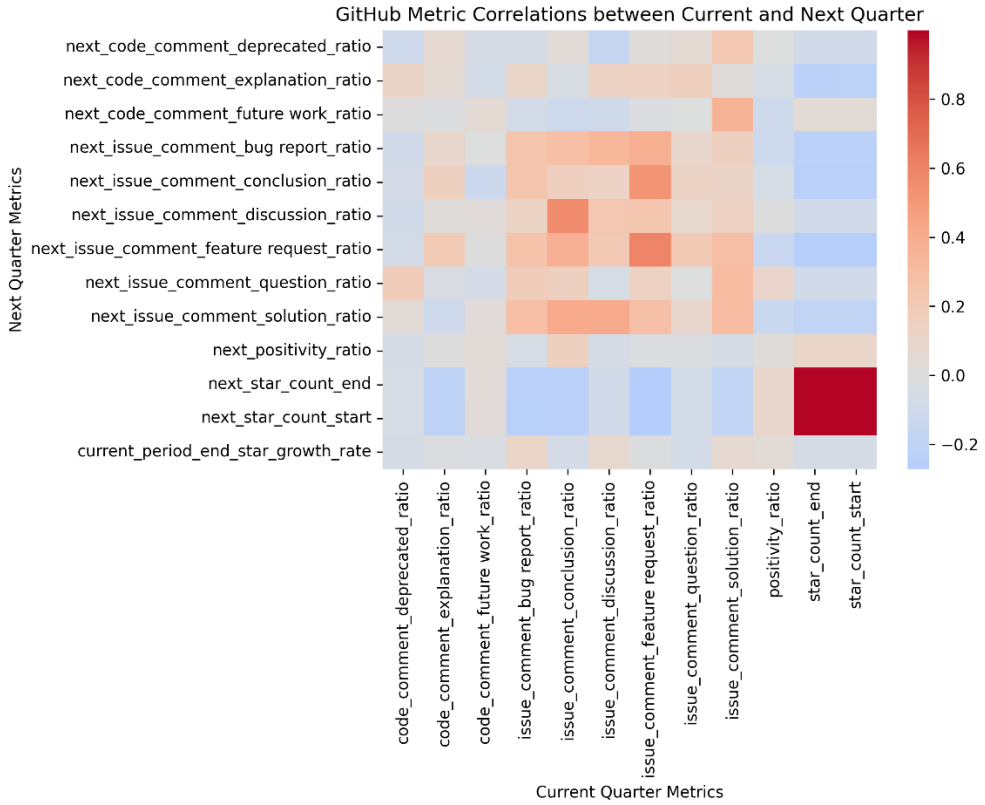


Feature Correlations

The correlations between each comment purpose and the future project popularity growth rate are as follows. The correlations are all quite weak, with purpose_explanation and purpose_question being the largest positive and negative correlations, respectively.



While the above chart shows correlations between features of individual comments, the following chart shows the correlations between feature aggregates grouped quarterly and by repository. Many interesting relationships exist. For example, feature requests are a positive feedback loop: having a relatively large proportion of feature requests this quarter is correlated with an even larger proportion of feature requests next quarter.



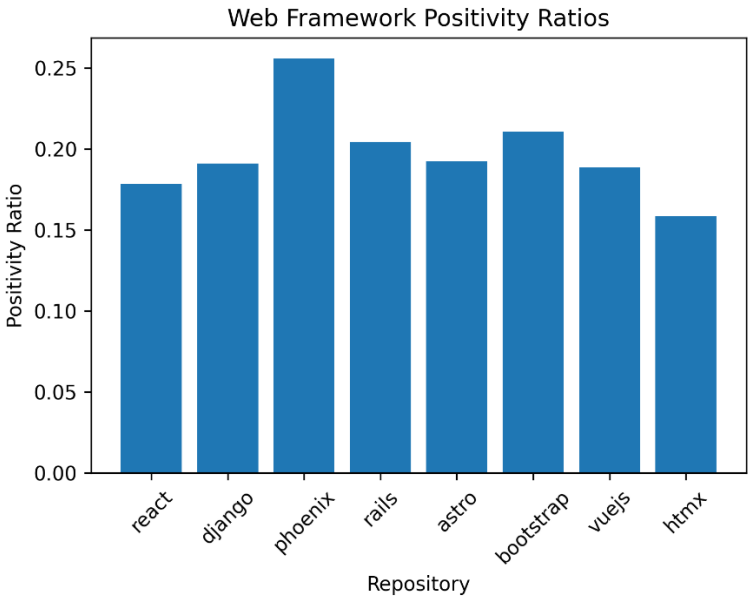
How does technical communication differ between projects?

For this portion of the analysis, the various ratios from the previous section were compared across web frameworks. The results indicate that communication style varies significantly across projects.

Comment Sentiment

The frequency of positive comments differs by up to 10 percentage points in web frameworks. The Phoenix framework has the most positive community (25%) while htmx has the least positive (15%). This is surprising to the author because Phoenix and htmx both have a reputation of being loved by developers. Further research is needed to discover the underlying cause for this discrepancy.

This difference is statistically significant, with a chi-square statistic of 59.10 and a p-value of 2.29×10^{-10}.

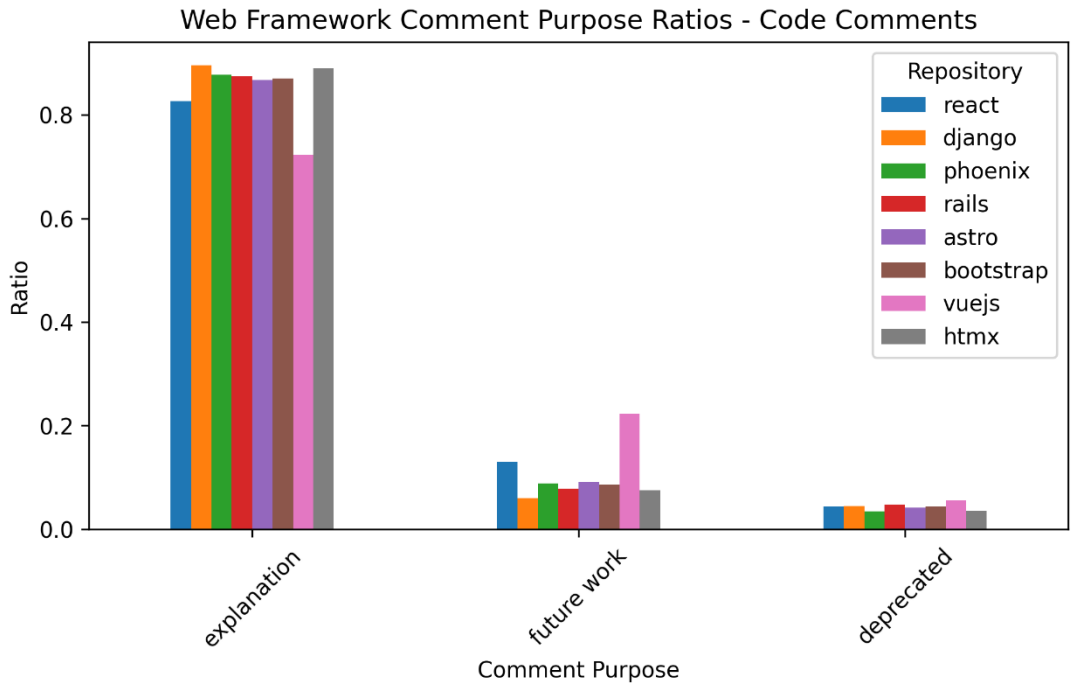


Comment Purpose

The distribution of code comment purposes does not vary much between repositories. The only major difference is that Vue.js tends to have a lot more “future work” comments than other repositories. This is surprising since Vue is a relatively mature project. Further research is needed to understand the reason for this difference.

Although small, the differences in code comment ratios are statistically significant, with a chi-square statistic of 78.50 and a p-value of 5.37×10^{-11}.

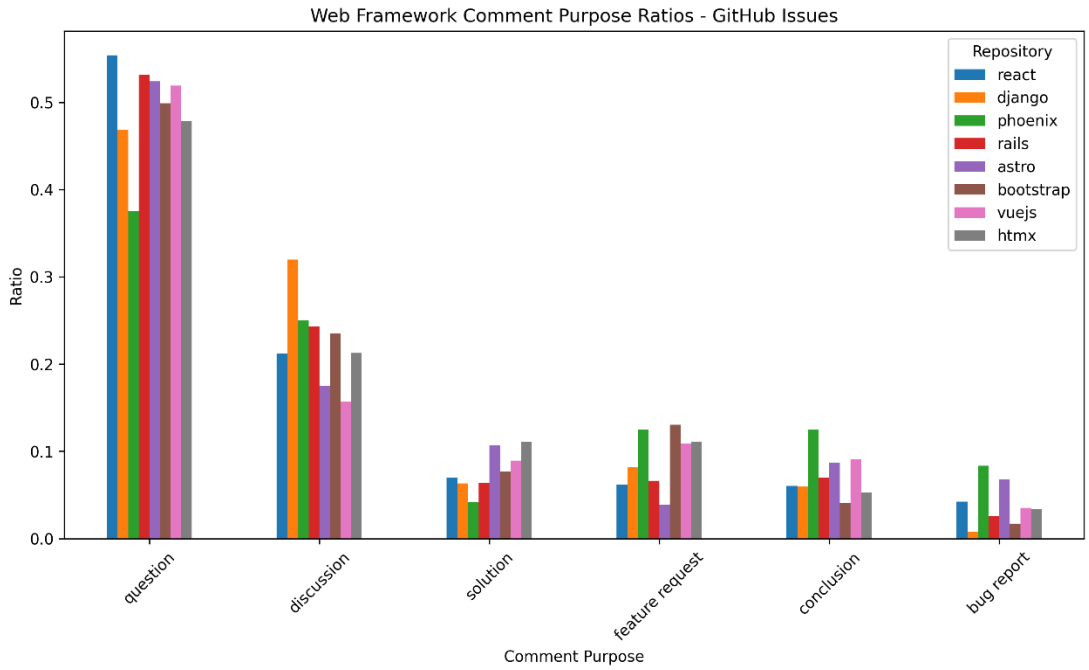
The purpose ratios in GitHub issue comments differ much more strongly. For example, React comments are by far the most likely to be questions. This may be due to React’s popularity. Since React has been the dominant frontend framework for many years, it likely attracts the most new, inexperienced developers, who are more likely to ask questions.



Also of note is Django's tendency to be an outlier in this regard. Django boasts an extremely small bug report rate and a high discussion rate. The reason why Django's community interacts so differently is uncertain but may be because Django is a relatively old and stable project. If Django developers are slow and deliberate in their changes, it seems reasonable that they would tend to have more open-ended discussions and fewer bugs.

Rails has an overall similar contour to Django (albeit less extreme). Rails has a similar design philosophy to Django – both prioritize developer experience and speed over all else. This design similarity seems to result in a similar communication style.

Phoenix has a high frequency of feature requests and bug reports. This implies that Phoenix is relatively new and immature compared to the other frameworks. This is corroborated by the author's anecdotal experience with the Phoenix framework, who found that it is missing many key features that are included with more mature frameworks like Django.



Once gain, these differences are statistically significant, with a chi-square statistic of 176.25 and a p-value of $9.51 \times^{10-21}$.

How does technical communication affect the growth of software projects?

The linear regression model for predicting log star growth rate achieved an R^2 of 0.066 with the following features. The coefficients are relative to the "purpose_solution" feature, which was dropped to reduce multicollinearity. Features with a p-value greater than 0.05 are highlighted.

Feature	Coefficient	P-value
const	0.2321	0.000
sentiment_class_POSITIVE	-0.0123	0.106
purpose_bug_report	-0.0215	0.506
purpose_conclusion	-0.0250	0.331
purpose_deprecated	0.1193	0.000
purpose_discussion	-0.0312	0.120
purpose_explanation	0.2249	0.000
purpose_feature_request	-0.0174	0.453
purpose_future_work	0.2175	0.000
purpose_question	-0.0309	0.093

In other words, code comments explain about 6.6% of the variation in software project popularity growth. Although this R^2 value is quite small, it is still statistically significant. In the domain of social interaction, low R^2 values are not surprising because human behavior is influenced by a huge number of factors. In fact, this R^2 is higher than the author expected to find. Even a small effect size, compounded over years of project growth, can have a measurable effect on project visibility and success.

According to the model, the largest positive effect on repository growth comes from adding new explanatory comments to a codebase. This is closely followed by adding future work comments, then by deprecation comments. No other feature had a significant effect.

Surprisingly, sentiment does not have a significant impact on growth rate. This was unexpected since sentiment has been found to have statistically significant effects in similar studies, as explained in the introduction.

Conclusions

The conclusions of this study are given in the context of the initial research questions.

What does technical communication look like?

In codebases, 80% of technical communication is explanatory code comments, with the remainder consisting of notes of future requirements and deprecated functionality. In GitHub Issues, 50% of technical communication is questions, 20% is discussions, with the remainder consisting of bug reports, solutions, and conclusions. In both code and issue comments, fixing problems is the most common topic of discussion.

How does technical communication differ between projects?

Communication types differ significantly between projects. Each software project has a distinct culture with different values, resulting in different communication styles and strategies. Out of all web frameworks analyzed, Django and Phoenix were the biggest outliers. Django focuses on reliability and discussion; Phoenix has the most bug reports and feature requests by a wide margin.

How does communication affect the growth of software projects?

The results from the linear regression model indicate that the purpose of comments explains about 6.6% of project growth. Explanatory code comments have the largest positive effect, with future work comments as a close second. In short, the most effective way to influence project growth with comments is to simply write code, explain what it does, and explain what it will need to do later.

However, one should not assume that the relationship between comments and growth is casual. Confounding variables were not eliminated during this study and are likely to exist. However, the wide range of programming languages is not likely to be one of these confounding factors considering that previous studies have been unable to find a link between programming languages and bug frequency (Berger et al., 2019).

Future Work

Many opportunities for future research exist in this space. As noted previously, more work is needed to understand the underlying causes for why different software projects have different comment purpose distributions. To gain deeper insight, one could expand beyond comments and analyze the actual source code of these projects. Code analysis will require some changes in methodology, but prior research has shown that natural language processing techniques such as Latent Dirichlet Allocation work well on computer code (Linstead, Lopes, and Baldi, 2008).

Furthermore, the dataset created for this project has significant untapped potential. For example, while this project focused mainly on predicting future project popularity, the same techniques could be used to determine the effect of technical communication on future bug report frequency or future sentiment. This project's dataset lays the groundwork for many potential future research projects.

Works Cited

B. Yang, X. Wei and C. Liu, "Sentiments Analysis in GitHub Repositories: An Empirical Study," 2017 24th Asia-Pacific Software Engineering Conference Workshops (APSECW), Nanjing, China, 2017, pp. 84-89, doi: 10.1109/APSECW.2017.13.

E. D. Berger, C. Hollenbeck, P. Maj, O. Vitek, J. Vitek, "On the Impact of Programming Languages on Code Quality: A Reproduction Study," ACM Trans. Program. Lang. Syst., vol. 41, no. 4, art. 21, Oct. 2019, doi: 10.1145/3340571.

E. Linstead, C. Lopes and P. Baldi, "An Application of Latent Dirichlet Allocation to Analyzing Software Evolution," 2008 Seventh International Conference on Machine Learning and Applications, San Diego, CA, USA, 2008, pp. 813-818, doi: 10.1109/ICMLA.2008.47.

M. L. Siddiq and J. C. S. Santos, "BERT-Based GitHub Issue Report Classification," in The 1st Intl. Workshop on Natural Language-based Software Engineering (NLBSE'22), Pittsburgh, PA, USA, May 21, 2022, pp. 1-4, ACM, New York, NY, USA, doi: 10.1145/3528588.3528660.

M. Mittal, A. Sureka, "Process Mining Software Repositories from Student Projects in an Undergraduate Software Engineering Course," ICSE Companion 2014: Companion Proceedings of the 36th International Conference on Software Engineering, New Delhi, India, May 2014, pp. 344-353, doi: 10.1145/2591062.2591152.

R. Jongeling, P. Sarkar, S. Datta, A. Serebrenik, "On negative results when using sentiment analysis tools for software engineering research," Empirical Software Engineering, vol. 22, no. 5, pp. 2543-2584, 2017, doi: 10.1007/s10664-016-9493-x.

V. Sinha, A. Lazar and B. Sharif, "Analyzing Developer Sentiment in Commit Logs," 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR), Austin, TX, USA, 2016, pp. 520-523.

Source code used for this project: https://github.com/joshmoody24/github-nlp