Perceived Language Complexity in GitHub Issue Discussions and Their Effect on Issue Resolution

David Kavaler*, Sasha Sirovica*, Vincent Hellendoorn*, Raul Aranovich†, Vladimir Filkov*
University of California at Davis, USA
* Department of Computer Science † Department of Linguistics
{dmkavaler, sasirovica, vhellendoorn, raranovich, vfilkov}@ucdavis.edu

***Abstract*—Modern software development is increasingly collaborative. Open Source Software (OSS) are the bellwether; they support dynamic teams, with tools for code sharing, communication, and issue tracking. The success of an OSS project is reliant on team communication. E.g., in issue discussions, individuals rely on rhetoric to argue their position, but also maintain technical relevancy. Rhetoric and technical language are on opposite ends of a language complexity spectrum: the former is stylistically natural; the latter is terse and concise. Issue discussions embody this duality, as developers use rhetoric to describe technical issues. The style mix in any discussion can define group culture and affect performance, e.g., issue resolution times may be longer if discussion is imprecise.**

Using GitHub, we studied issue discussions to understand whether project-specific language differences exist, and to what extent users conform to a language norm. We built project-specific and overall GitHub language models to study the effect of perceived language complexity on multiple responses. We find that experienced users conform to project-specific language norms, popular individuals use overall GitHub language rather than project-specific language, and conformance to project-specific language norms reduces issue resolution times. We also provide a tool to calculate project-specific perceived language complexity.

I. Introduction

Broadly speaking, short words are best...

Winston Churchill

Written and spoken language is part and parcel of modern software development. Millions of developers use GitHub to code amazing new software. They collaborate and communicate about the code, underlying design and arising issues, creating multi-layered socio-technical communities. Those OSS developers come from different geographical and social backgrounds. They have differing *social and cultural identities*, informed by their socioeconomic status, cultural background, and myriad other factors. This influences how they talk, how they listen and how they understand written and spoken language.

Sociological theories tell us that community membership must be considered part of an individual's social identity, and OSS being analogous to a workplace, or a social activity, GitHub and OSS projects take on the roles of communities. In a sociolinguistic sense, GitHub and OSS projects frame the conversations and provide context. Like in other communities, these too have their own specific language idiosyncracies, composed of project-specific terms and related jargon. Dialing in the "correct" linguistic markers when talking to other developers in the project is a mark of identity as well as standing. It is natural to ask, then, to what extent are these linguistic markers apparent on GitHub and within specific projects? And do they impact productivity or effectiveness?

Answering such questions is contingent on being able to measure a) community specific language idiosyncracies, or even norms, if they exist, and b) the amount of departure from them in any given written or spoken language. There are many different characteristics of language that can be measured. One which is particularly salient in communication and collaboration is understandability, or simplicity of the language used. This can be operationalized through various information theoretic measures, especially entropy, as we do later in the paper. Then, community linguistic norms can be studied by looking at corpora of text written or spoken by members of that community. Models built from those corpora can then define the *community language*. Once a linguistic norm is found, we can use entropy-like measures to compare the departure of new text from the linguistic model of the normative language (built from existing corpora of community communication). This understandability as a function of both the speaker/writer's language use and the listener/reader's perception, will be referred to as the *perceived language complexity*. It is a notion of how understandable a writer's text is to a community of readers.

In GitHub projects, discussions on issues are a natural focus for a sociolinguistic study of community linguistic norms. Discussions on issues involve multiple developers, and often turn into back-and-forth conversations on the merits of the issue and potential solution(s). These conversations are a combination of rhetoric, where people present their arguments in a narrative style,[1] and technical arguments arising from the project particulars. With the abundance of projects on GitHub, issue discussions provide a rich data source for studying linguistic determinants for outcomes relevant to software engineering.

In this work, we study community specific language complexity and perceived complexity, measured by Shannon entropy and language models, from issue discussions in tens

[1] When we refer to rhetoric, we do not identify with the modern colloquial term, with potential negative connotations. We refer to rhetoric in the philosophical sense - as the art of argumentation, where one posits a premise and argues soundly.

978-1-5386-2684-9/17 © 2017 IEEE

ASE 2017, Urbana-Champaign, IL, USA
Technical Research

of the most starred and followed GitHub projects. The data consist of $90,722$ issues, comprising $456,669$ total posts. We explore the specific questions: is there evidence of a standard GitHub perceived language complexity? And does this carry to projects? Is there migration in perceived language complexity towards the norm? *I.e.*, do new project participants gravitate towards the project perceived language complexity norm? And what may drive this linguistic acculturation? What is the relationship, if any, between perceived language complexity and issue resolution latency? Remaining mindful of various confounds and threats to validity, we provide the following contributions:

- We collected a data set comprised of many issue discussions, including issue post text, user-related metadata, and constructed a social network defined by @*mentions* to other users within and outside the project to determine user project popularity defined by contributions in issue discussions.
- We use unique language modeling strategies to create a global GitHub language model, along with nested dynamic project-specific language models to identify and study perceived language complexity through language model entropy. We find that project-specific language models outperform the global GitHub language model (statistically significant), indicating that *project-specific terms and perceived language complexity exist*, and *some projects have more project-specific perceived language complexity than others*.
- Using our language models inside of regression models, we find that *users gravitate towards the project language norm* and *popular users (measured by the number of times they are @mentioned in issue discussions) are better represented by the GitHub global language model than project-specific language models*.
- We also find that *increased project perceived language complexity increases issue latency*.
- We provide a tool[2] to calculate perceived language complexity given a user's input and specified project.

This paper is an initial foray into language complexity analysis and its relationship to OSS and communities of practice in general. We hope to help bridge the two fields, providing a coarse study of the links between them.

The remainder of this paper is organized as follows: Section II presents theory and related work; Section III outlines our research questions; Section IV describes our data and methods; Section V presents our results and discussion; Section VI presents how to apply our results in practice; Section VII presents our conclusions.

II. Theory and Related Work

To understand the emergence of OSS project based linguistic communities, as well as the extent of their linguistic separation, we build theories that draw from a number of backgrounds. We first tackle the connection between language and community, by examining two sociolinguistic models: communities of practice, and speech communities, and then reason how GitHub communities fit in. Then, we turn our attention to the ongoing conflict between two language complexity paradigms, the technical and the rhetorical, as found in GitHub issue discussions. To model this conflict, we start from the anthropological theory of "*homo narrans*", which can explain the overly narrative structure of our communications, even technical ones. We also mention anthropological models of acculturation/assimilation that confound language use when communities merge. Further, we go over work on entropy as a measure of language complexity, which underlies the choices for our measures. Finally, we review research on collaboration during software development, which sets the larger context of building and maintaining relationships and communities in OSS.

A. Speech Communities and Communities of Practice

Sociolinguistics is the study of how societal aspects influence language, and the effects of language on society [1]. The two primary theoretrical frameworks regarding the influence of communities on language are those of *speech communities* and *communities of practice*. Here, we argue that GitHub projects do not fall solely into either theoretical framework, but require a blend of both.

A speech community (SC) describes a group of people who use language in a way that is mutually accepted among the group [1]. To belong to an SC, one must have communitative competence [2]; they must speak in a manner that is standard within the SC. SC members often speak with specialized terms or form jargon that is understood within the community [3]. Another way to express this is the idea of *norm conformity*, where the language norm is set by the community and individuals are expected (or aspire) to speak in a manner according to said norm.

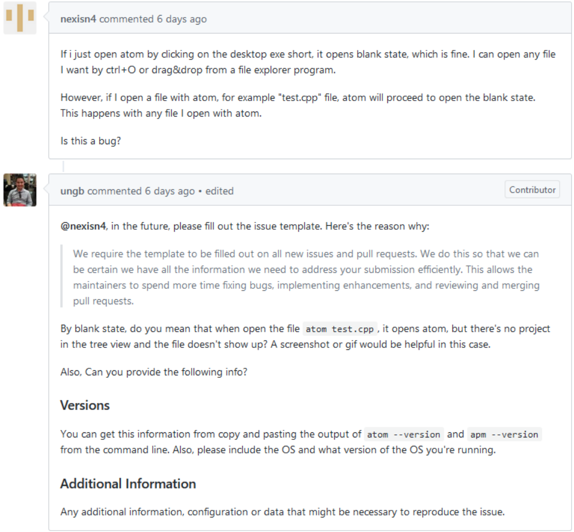
First coined by Wenger and Lave [4], the notion of a community of practice (CoP) is widely discussed and debated in the field of sociolinguistics. Broadly, a CoP can be described similarly to a SC - members use language in a specialized way that is defined by the group, and group members are expected to conform to the group language norm. The precise definition of this notion has been expanded and expounded over time by Wenger and many others [5], [6]. Wenger proposed multiple critical characteristics of a CoP, outlined by Holmes and Meyerhoff [7], contained in Table I[3].

Figure 1 is an example where a member of the out-group (an end-user) has an issue. They attempt to express their problem, but the contributing member (in-group) is unable to understand the description, and further asks the end-user to conform to project-specific norms (*i.e.*, fill out the standard issue report form)[4]. This is an example where a lack of community norm conformity causes confusion.

[2] https://github.com/normative-team/normative

[3] Note that all the characteristics listed in [7] can be applied to GitHub; we select the most relevant for brevity.

[4] Note that the in-group member seems to also be confused by the English language description, not just the lack of a standard issue report form.



nexisn4 commented 6 days ago

If i just open atom by clicking on the desktop exe short, it opens blank state, which is fine. I can open any file I want by ctrl+O or drag&drop from a file explorer program.

However, if I open a file with atom, for example "test.cpp" file, atom will proceed to open the blank state. This happens with any file I open with atom.

Is this a bug?

ungb commented 6 days ago • edited

@nexisn4, in the future, please fill out the issue template. Here's the reason why:

> We require the template to be filled out on all new issues and pull requests. We do this so that we can be certain we have all the information we need to address your submission efficiently. This allows the maintainers to spend more time fixing bugs, implementing enhancements, and reviewing and merging pull requests.

By blank state, do you mean that when open the file `atom test.cpp`, it opens atom, but there's no project in the tree view and the file doesn't show up? A screenshot or gif would be helpful in this case.

Also, Can you provide the following info?

Versions

You can get this information from copy and pasting the output of `atom --version` and `apm --version` from the command line. Also, please include the OS and what version of the OS you're running.

Additional Information

Any additional information, configuration or data that might be necessary to reproduce the issue.

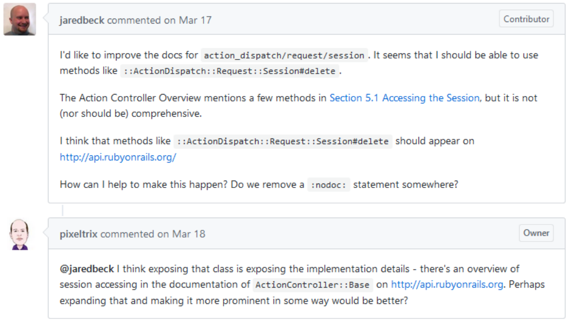
Fig. 1. Issue thread 14357 from the *atom/atom* project.

TABLE I
DIFFERENCES BETWEEN A SPEECH COMMUNITY AND A COMMUNITY OF PRACTICE [7].

Speech Community	Community of Practice
Shared norms and evaluations of norms are required.	Shared practices are required.
Shared membership may be defined externally.	Membership is internally constructed.
Nothing to say about relationship between an individual's group and personal identities.	Actively constructed dependence of personal and group identities.
Non-telelogical.	Shared social or instrumental goal.
Nothing to say about maintenance or (de)struction of boundaries between categories.	Boundaries are maintained but not necessarily defined in contrasts with outgroups.
Acquistion of norms	Social process of learning.

Research in CoPs initially focused on the classroom and businesses in which students and employees are physically co-located. Important to us is the work of Dubé *et al.*, who argued that members of a CoP do not have to be physically co-located, but can form a "virtual community of practice" [8] with the same attributes as a standard CoP. In addition, Kleinnijenhuis *et al.* discuss Networks of Practice, a further refinement of sociolinguistic ideas *w.r.t.* community organization, specifically for online communities [9].

Table I shows partially overlapping characteristics between SCs and CoPs in each row. Those within the same project share practices, as all members are on GitHub. However, membership is not entirely internally constructed. Llamas *et al.* [10] present an example of a CoP within the workplace, saying that individuals regularly engage in scheduled social practices (*e.g.*, business mettings), and *mutually define themselves* as CoP members. Eckert [11] describes SCs as having membership defined by "broad and fundamental social categories", such as socioeconomic status, age, and ethnicity.



jaredbeck commented on Mar 17

I'd like to improve the docs for `action_dispatch/request/session`. It seems that I should be able to use methods like `::ActionDispatch::Request::Session#delete`.

The Action Controller Overview mentions a few methods in Section 5.1 Accessing the Session, but it is not (nor should be) comprehensive.

I think that methods like `::ActionDispatch::Request::Session#delete` should appear on http://api.rubyonrails.org/

How can I help to make this happen? Do we remove a `:nodoc:` statement somewhere?

pixeltrix commented on Mar 18

@jaredbeck I think exposing that class is exposing the implementation details - there's an overview of session accessing in the documentation of `ActionController::Base` on http://api.rubyonrails.org. Perhaps expanding that and making it more prominent in some way would be better?

Fig. 2. Issue thread 28464 from the *rails/rails* project.

In essence, membership in a CoP is much more structured, where members of the in-group determine who else is a member, while membership in an SC lies in factors that the group itself may not control, *e.g.*, cultural background or self-identification. The idea of one's social identity is of extreme importance in the field of sociolinguistics, where it is well-known that social identity is informed by, *e.g.*, one's workplace and social activities, and contributes heavily to language use and variety [7].

On GitHub, developers of a project fall heavily into the CoP characteristic of internally constructed membership, analogous to a workplace. The community of users, we argue, is more like that of an SC than that of a CoP, due to their ability to self-identify with the project through their use of the product. One cannot unilaterally separate the users from the developers of an OSS project, as they are a vital part of the ecosystem. In fact, our data shows that many issues are not initially posted by project members (*i.e.* committers), but by general users. Thus, in terms of membership, GitHub projects share aspects of an SC and a CoP. Due to the inclusion of the users as part of the project ecosystem, one can make a similar argument for the remaining listed characteristics. As one can see, GitHub projects are not solely classified by either theoretical framework; the framework necessary to describe the complex interactions in GitHub projects is a feedback between both the ideas of SCs and CoPs.

B. *Homo Narrans*

GitHub issues serve a direct purpose within the community: to, *e.g.*, raise concerns or report bugs in the related software. However, their form is that of discussion; a user posts an issue, and anyone on GitHub is free to post responses and express their opinion, as in a message board. This provides opportunities for rhetorical communication. An example of this can be seen in Figure 2. The first poster posits an issue ("I'd like to improve the docs...") and suggests a solution ("I think that methods like ... should appear on ..."). Another poster responds with a counter-idea, and the conversion continues (not shown).

We can imagine that each developer resolves an internal conflict when posting on these discussions, a conflict about how technical, or terse, vs. rhetorical, their post should be.

The following theory offers an explanation for how such a conflict is likely to be resolved, in a population of developers.

It is accepted that technical language (*e.g.*, within a technical manual) is often terse, concise, and prone to the inclusion of jargon when compared to natural language [3], [12]. One may believe that rhetorical language is likewise comparatively terse and concise. However, in his seminal work, Fisher [13] argues (in summary by Hauser [14]) (specifically *w.r.t.* rhetorical communication) that "all humans are definitively story tellers; they are of the species *homo narrans*... their communication assumes the basic form of stories". Essentially, Fisher argues that, by nature, all communication must be seen in the lens of narration, and thus even in rhetorical language the argumentator forms a narrative. Although GitHub issues contain significant amounts of technical language, their function as rhetorical communication separates them from being terse and concise. GitHub discussions are a push-and-pull effort between the push to use technical language that is relevant to the project, and the pull to provide a narrative which frames one's position through rhetoric. Thus, GitHub discussions may lend their language classification to be somewhere between purely technical language and narrative language.

C. Acculturation and Assimilation

Acculturation and assimilation are the processes that result from the merger of fusion of two cultures [15], typically a majority one and a minority one [16]. Both are two way processes, but the former term describes an outcome in which both cultures change, and the minority one retains cultural distinctions like language, cuisine, *etc*. The latter, assimilation, is when the identifying characteristics of the minority culture gradually get dissolved into the majority culture. Assimilation reduces the communication overhead and conflicts inside a group [17], while acculturation exemplifies diversity [15].

Linguistic acculturation in particular is the process of culture phenomena influencing the language used. Such acculturation can happen fairly rapidly, and has been studied in both immigrants who learn a second language [18] and communities of practice [19]. Communities of practice, especially those on the social web, are good examples of linguistic acculturation standing in for performance and status based metrics. Thus, adjusting to the local culture and language fairly rapidly can be seen as a successful migration into a community.

In the context of developer communities on GitHub, developers who join new projects may be able to acculturate or even assimilate if the project culture is sufficiently dissimilar from their own. Prior results on work/talk culture in OSS communities [20], [21] points in the direction that strong incompatibility of an individual and a project culture may lead to failure to acculturate, and subsequent departure from the project.

D. Entropy and Language Complexity

Language models, generally, attempt to predict the next word in a text given the word's context (normally the preceding n words). The metric by which language models are generally evaluated is *entropy* (described precisely in Section IV-C). Kolmogorov [22] presented a formulation of the connection between entropy and complexity in terms of predictability, which has been discussed in length [23], [24].

There has been work in using entropy as a description of language complexity and style. Kontoyiannis [25] studied hwo entropy represents the complexity of language in the literary sense. They computed entropy per character of the Bible and the writing of Jane Austen and James Joyce using a specialized language model that utilizes a sliding window of context. They found that a larger context window leads to a lower entropy estimate for the Bible and a higher estimate for Jane Austen and James Joyce novels. Repetitive language style, as in the Bible[5], has lower entropy and is easier to understand and read[6], while the style of, *e.g.*, James Joyce seems more complex and harder to read.

In this work, we hypothesize that norm conformity (as, *e.g.*, members of the out-group are assimilated to the in-group) can be measured in terms of *perceived language complexity*. Thus, we define the following:

A community's language is defined by sociolinguistic norms set by members of the group. In this work, we refer to the GitHub community (global) and nested project-specific (local) community languages. These community languages are represented by a language models.

Perceived language complexity is the distance between a community's language and a given text. This is quantified using the (cross-) entropy distance (Section IV-C).

As perceived language complexity increases, the understanding of the text by members of the community will decrease.

Additionally, we study the difference between global and local language, represented by separate language models. Juola and Baayen [26] found success in using cross-entropy to settle authorship disputes. This lends creedance to our method of comparing global GitHub language model entropy to project-specific language model entropy in describing project differences. Kwon *et al.* [27] proposed a model for representing narrative complexity, and use entropy as a quantitative measure to describe this narrative complexity. As we state, based on Fisher's work [13], that rhetoric is really a form of narrative, the decision of Kwon *et al.* to use entropy as a measure of narrative complexity supports our approach.

E. Collaborative Work in Software Engineering

Dabbish *et al.* [28] studied the effects of transparency (*e.g.*, the ability to see everyone's contributions) on GitHub projects and found that project members use this transparency to make rich inferences about others' goals. This lends creedance to

[5] They state that it is a well-known fact in linguistics that, excluding proper names, there are only about 500 roots in the Bible

[6] Though the thoughts and feelings invoked as reported by human subjects can be much deeper than that of a less repetitive style.

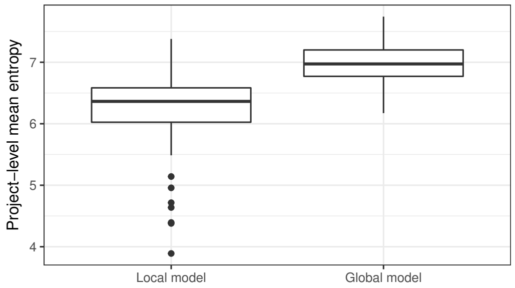


Fig. 3. Project-level mean post entropy for all projects under study (paired Welch two sample t-test and paired Wilcoxon rank sum test $p < 0.001$).

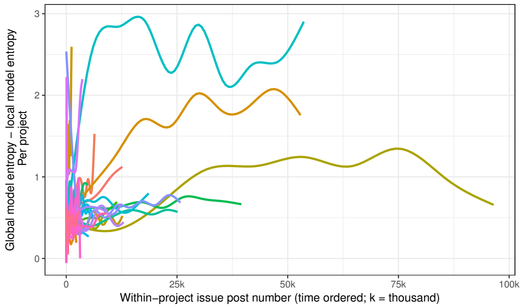


Fig. 4. Difference (*i.e.*, separation) between global and local model entropy per observed post, per project. Smoothed by cubic regression spline (shrinking).

the existence of a CoP structure; CoP membership is internally constructed, and in-group members use transparent details to determine who should be a member of the in-group. Tsay *et al.* [29] examined pull-request discussions and the effects of various measures (*e.g.*, test inclusion, prior social interaction of submitter, *etc.*) on acceptance. Blincoe *et al.* [30] found that popular users attract their followers to new projects. This finding is of interest as new developers to projects are likely from the out-group, which means they will need to conform to local linguistic norms in order to communicate effectively. Xuan *et al.* [20], [21] studied work/talk patterns in OSS projects and found that these patterns eventually converge in time, or the users with dissimilar patterns will leave the project. Similar to ours is the work of Yu *et al.* [31], who examined determinants of pull request latency. Our work concerns issues in general, not just pull-requests, and focuses on language as a predictor rather than other measures. Given the strong theoretical backing in sociolinguistics, there is a distinct lack of work on quantiative linguistic analysis of communication and collaboration traces (*e.g.*, issues) and effects on software engineering outcomes. We present an initial look into this domain.

III. Research Questions

Guided by the discussion in our theory (Section II), we investigate a number of research questions.

The GitHub community is organized into projects, or its sub-communities. SC and CoP theory both state that there exist community language norms, as effective communication relies on expressing oneself in a manner that resonates with the community. Thus, we hypothesize that project-specific language differences will work to separate the project language, analogous to, *e.g.*, software engineering, from the larger GitHub community language, analogous to, *e.g.*, the Computer Science academic community. The theory also suggests that given that GitHub issue discussions are a complex combination of rhetoric and technical terms, their perceived language complexity should lie somewhere in the spectrum between rhetorical narratives (*e.g.* novels) and technical manuals.

RQ 1: Is there evidence for a standard for the GitHub community language? Does this also carry over to projects, *i.e.*, is there evidence for a project-specific language?

As posited by SC and CoP theory, those who wish to communicate effectively within a community must do so in a way that is guided by the community language. Thus, we expect that the perceived language complexity of posts will migrate, in time, toward the project norm.

RQ 2a: Is there migration in perceived language complexity, over time, toward the project norm?

As OSS projects are dynamic, there are always users coming and going. As posited by SC and CoP theory, new members of the community will adopt the community-specific language in order to communicate effectively and signal themselves part of the in-group. Thus, as a corollary of research question 2a, we hypothesize that new contributors (be they end-users or developers) to issue discussions are better represented by the global language than the local language, and will eventually conform to the local norm. In addition, we hypothesize that experienced or popular users will be more in-tune to the project, and that their perceived language complexity will reflect this.

RQ 2b: Do users (popular or experienced) conform to their associated project language? And does perceived language complexity influence popularity?

Finally, we seek to identify any effects of perceived language complexity on outcomes of particular interest to software engineering. Specifically, if the language in a post or discussion is more uncommon given the global or local norm (i.e., less understandable to the community), we hypothesize that issue resolution latency will increase.

RQ 3: What is the relationship, if any, between perceived language complexity and issue resolution latency?

IV. Data and Methods

A. Issues

GitHub projects have issue trackers with a rich feature set, including ticket labeling, milestone tracking, and code tagging. For each project on Github, individuals can open up an issue thread where others can comment and discuss a specific issue.

GitHub contributions are centered around *pull-requests*. Users who wish to add a feature can submit a pull-request to be incorporated into the main project. GitHub treats pull-requests as a subtype of issues; thus, we consider both ordinary issues and pull-requests in our data.

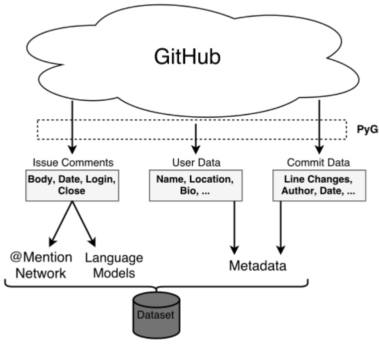


Fig. 5. Overview of dataset formation.

We queried GitHub's public API using PyGithub.[7] Our data is a sample of 48 projects from the top 900 most starred and followed projects.[8] The number of stars and followers are proxies for project popularity, and can identify projects likely to contain enough issues to build robust language models. Figures 1 and 2 depict issue threads. For every post in an issue thread, we gathered the following information: date of post, post body, login of poster, and issue closing time. Post bodies were used to extract text-related metadata and to build our language models. In addition to examining comments we also collected a variety of metadata from projects. This included commit-related metrics: number of lines added and deleted, date of commit, commit author; and user data: full name, time they joined Github, and location. Figure 5 presents a graph of our data pipeline.

B. Social Networks

We constructed a social network for each project using *@mentions* in their issue comment threads, similar to Zhang *et al.* [32]. @mentions are commonly used to either reference or reply to someone else in a discussion. Figures 1 and 2 show examples of @mentions in use.

In our networks every edge (u, v) is represented by user u @mentioning v somewhere in their post. In addition, we track changes in network measures over time to construct time-varying networks, used to more accurately represent a user's attributes at the time of posting. The key metric extracted from the social network is *indegree*, used as a measure of within-project user popularity. Additional metrics calculated from the networks include outdegree, betweenness, and degree centrality, but were not used in our models due to multicollinearity.

C. Language Models

Measuring perceived language complexity relies on two components: the training data and the language model. The choice of training data (or corpus) determines what sequences of tokens (*e.g.*, words, punctuation) the language model will deem likely or unlikely at test time, whereas a good choice of language model should capture useful patterns and ensure that rare or novel sequences are not assigned zero probabilities. This combination allows for a distance metric between two corpora in terms of the (left-to-right) predictability of the tokens t_i in the test corpus, given a language model LM trained on the training corpus:

$$d_{LM}(train, test) = \prod_{i=1}^{|test|} p_{LM}(t_i|t_1 \cdots t_{i-1})$$

In practice, instead of a product over probabilities, we use the average of (binary) log-probabilities, which yields an *entropy* score that reflects how many bits of information the average token in the test corpus has *w.r.t.* the training corpus:

$$d_{LM}(train, test) = \frac{1}{|test|} \sum_{i=1}^{|test|} -\log_2(p_{LM}(t_i|t_1 \cdots t_{i-1}))$$

As mentioned, entropy is our measure of perceived language complexity.

1) Choice of Corpus: To create a representation of the GitHub community language, we must create a corpus that is representative of GitHub as a whole. To accomplish this, for each project, we construct a corpus consisting of *all posts in all other projects*. Thus, each project has a corresponding model that is representative of the GitHub community as a whole; we call this the *global model*. Note that each project has a different global model, as we must be careful not to overlap project-specific community language with our representation of the whole GitHub community language. On average, each project's global model is trained on $\sim$ 25M tokens.

To create a representation of project-specific languages, we use each project's associated global model as a *static representation*, and chronologically update a project-specific language model with each observed post from a given project. This yields an evolving model that captures project-specific idiosyncracies in the presense of a global norm; we call this the *local model*. Each local model is a *dynamic* mix-model, mixing a static global model with a dynamic local model. As this is an atypical use-case for conventional language models, we implemented it using SLP-Core [33], a library designed for mixing and dynamically updating language models.

2) Choice of a Model: RNN/LSTM models [34] are some of the most powerful language models available, but our cross-project setting is such that it would be computationally infeasible to use these models.[9] Furthermore, updating the model per post is poorly supported by these models. Instead, we use n-gram language models, which are proven as powerful yet simple tools to capture much of the complexity of both natural language and source code, even rivaling LSTMs in the latter. In addition, n-gram models are much faster when used for prediction than their neural network counterparts, allowing us to build a tool for fast predictions (Section VI).

An n-gram model uses the counts of sequences in the training data. It assigns a conditional probability to a token given the $(n-1)$ preceeding tokens by dividing the count of

[7] https://github.com/PyGithub/PyGithub

[8] We sampled 50 projects, but with replacement, leading to 48 projects. 50 projects is the upper bound for what is reasonable to process within a day.

[9] In total, we train and test on $\approx$ 1.2 billion tokens of data, twice.

TABLE II
POSTER MEAN ENTROPY MODELS. USER-LEVEL OBSERVATIONS.

	Dependent variable:	
	Poster mean post entropy (local)	Poster mean post entropy (global)
Log mean # tokens in posts by user	−1.262***	−0.789***
Log # of posts by user	−0.0003***	0.00001
Log mean # URLs in posts by user	−0.300***	−0.129***
Log mean # non-English words in posts by user	1.287***	1.016***
Mean Flesch Reading Ease of posts by user	−0.002***	−0.002***
Log poster indegree (as of last post)	**−0.003**	**−0.047*****
Log # additions + deletions by poster (as of last post)	**0.003**	**0.019*****
Log # followers of poster (as of last post)	−0.014***	−0.033***
Log # public repos for poster (as of last post)	0.018***	0.023***
Log # public gists by poster (as of last post)	0.010*	−0.013***
Log poster's GitHub age (hours) (as of last post)	**−0.015*****	**−0.013*****
Log poster's project age (hours) (as of last post)	**−0.017*****	**−0.020*****
Intercept	7.084***	6.713***
R^2	0.259	0.238
Note:		*p<0.05; **p<0.01; ***p<0.001

the whole sequence (length n) by the count of the context sequence (length $n - 1$), effectively making a Markovian assumption about word distribution. The model is generally *smoothed* by combining this maximum likelihood estimate probability with those returned by repeating the process for a shorter (*e.g.*, length $n-2$) context, until reaching the empty context, where the model just uses the vocabulary rate of the token. Various smoothing methods propose different rules to mix longer contexts with shorter ones depending on how "confident" they are in the longer context. We compared various n for various smoothing methods and found little difference in entropies scores across all 3-5-gram models. Thus, we use a 4-gram Witten-Bell smoothed model; a general purpose smoothing approach that has historically been used in text compression [35]. Finally, before training each model, we extract a vocabulary from the training data and replace all tokens seen fewer than 10 times with a generic "unknown" token in the train and test data, and replace all well-formatted code segments with a special "code" token. This standard preprocessing step prevents the models from having to predict completely novel or very rare words at test time. On average, the vocabulary used for each model spans $36,164$ tokens (standard deviation 796 tokens).

D. Regressions

To seek answers for our research questions, we use ordinary least squares (OLS) linear regression. This allows us to inspect the relationship between our response (*dependent variable*) and our explanatory variables of interest (*predictors* or *covariates*, *e.g.*, user age), under the effects of various *controls*.

When performing regression modeling, one is not only interested in standard "goodness-of-fit" measures (*e.g.*, R^2), but also the results of model diagnostics [36]. In OLS regression, R^2 literally measures the percentage of variance captured by a model. However, a "low" R^2 value alone does not mean that the model cannot be inferred from, or that a model is somehow "incorrect" [37], [38], [39], [40]. For example, if the variance in the dependent variable is large (as is the case for our data), a "low" R^2 can mean a great deal, as even a small percentage of a large variance is meaningful.[10] When searching the space for the best model, we tried many model types, including Poisson and Quasi-Poisson generalized linear regression, and mixed-effects models. In all cases, we could construct a model with a much higher R^2 value (from 60% to 70%); however, these models do not pass diagnostics; they had a multitude of problems, including heteroscedasticity, autocorrelation of residuals, conditionally non-normal errors, *etc*.

Most important is that our models meet the assumptions of the given regression approach, as indicated by model diagnostics. We take great care to make sure that our models pass these diagnostics, and thus can be inferred from, rather than providing artificially inflated R^2 with incorrect models. When appropriate, we employ *log* transformations to stabilize the variance and improve model fit [41]. We remove variables that introduce *multicollinearity* measured by *variance inflation factor* > 5, as multicollinearity reduces inferential ability [41]. We control for many potential confounds through our controls, and make a best-effort to build models that are statistically robust, in spite of what may be considered a "low" R^2 value. Our models can be found in Tables II, III, and IV and are discussed in Section V.

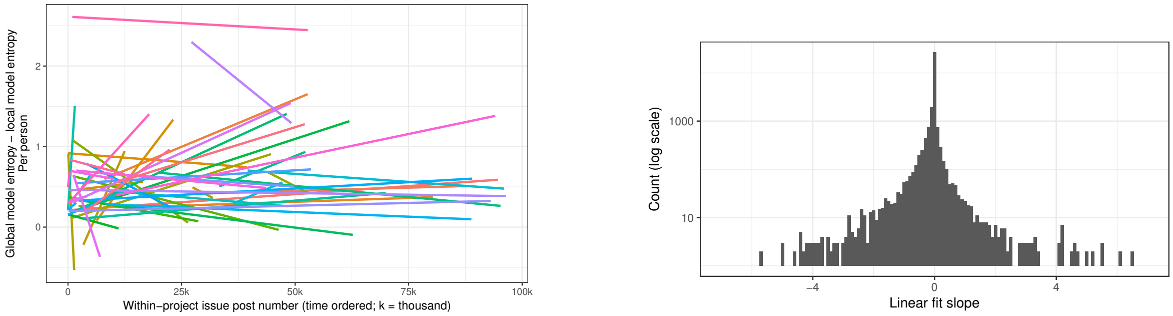
V. RESULTS AND DISCUSSION

In the models in Tables II, III, IV, our variables of interest are listed in bold; other variables act as controls for confounds, *i.e.*, post readability, measures of time, and code contributions. All project factors are omitted for brevity.

RQ 1: Is there evidence for a standard for the GitHub community language? Does this also carry over to projects, *i.e.*, is there evidence for a project-specific language?

Recall from Section IV that a local model learns to adjust over time to the community language; a global model is static. As shown in Figure 3, the local language model significantly outperforms the global model for each project ($p < 0.001$). The mean entropy for global models is 6.98 bits; 6.15 bits for local models (Cohen's d 1.689 [42] for the difference). We compare these results to that of n-gram language models built

[10] The raw value of explained variance is then large.



(a) Linear fits for per person per post difference between global and local model entropy over time (50 people with more than 50 total posts each sampled to reduce visual clutter).

(b) Histogram of slopes following the logic in Figure 6a for people with at least 2 posts. Bins of width 0.1. Not sampled.

Fig. 6. Plots indicating changes in language complexity over time for each person.

TABLE III
POSTER INDEGREE MODEL. USER-LEVEL OBSERVATIONS.

	User indegree (as of last post)
Mean entropy of posts by user (local)	**0.026*****
Mean entropy of posts by user (global)	**−0.030*****
Log mean # URLs in posts by user	0.015**
Log mean # non-English words in posts by user	0.041***
Mean Flesch Reading Ease of posts by user	−0.001***
Log poster outdegree (as of last post)	0.777***
Log # additions + deletions by poster (as of last post)	0.030***
Log # followers of poster (as of last post)	0.010***
Log # public repos for poster (as of last post)	−0.011***
Log # public gists by poster (as of last post)	−0.004
Log poster's GitHub age (hours) (as of last post)	**0.018*****
Log poster's project age (hours) (as of last post)	**0.071*****
Intercept	−0.131***
R^2	0.685
Note:	*p<0.05; **p<0.01; ***p<0.001

TABLE IV
ISSUE RESOLUTION LATENCY MODEL. ISSUE-LEVEL OBSERVATIONS.

	Log issue resolution latency (hours)
Mean issue entropy (local)	**0.116*****
Mean issue entropy (global)	**−0.008**
First post entropy (local)	**0.023***
First post entropy (global)	**0.004**
Log # URLs in first post	0.175***
Log # non-English words in first post	0.220***
Flesch Reading Ease of first post	−0.0001
Log poster indegree	0.001
Hour of day (first post)	−0.012***
Monday (factor) (first post)	−0.244***
Tuesday (factor) (first post)	−0.311***
Wednesday (factor) (first post)	−0.212***
Thursday (factor) (first post)	−0.209***
Friday (factor) (first post)	−0.099*
Saturday (factor) (first post)	0.125*
Log # additions + deletions by poster	−0.073***
Log # followers of poster	0.027***
Log # public repos for poster	−0.011
Log # public gists by poster	0.009
Log poster's GitHub age (hours)	0.016*
Log poster's project age (hours)	0.054***
Log number of posts in issue	0.572***
Log number of unique posters in issue	1.758***
Log ordered time of first post (ordinal)	−0.096***
Intercept	−1.207***
R^2	0.254
Note:	*p<0.05; **p<0.01; ***p<0.001

on technical manuals written in English, reported by Hasan and Ney [43] to lie between 5.58 and 4.85 bits, depending on model type, and those built on English, reported to be ~ 8 bits by Hindle *et al.* [44] and Tu *et. al* [45] on a combination of the Brown and Gutenberg corpora[11]. Note that the Brown and Gutenberg corpora may be considered more terse and concise than a corpus consisting of, *e.g.*, James Joyce novels, and still has higher entropy than our models.

We hypothesized in Section II that the post entropy of GitHub issues would lie somewhere between that of technical manuals and narrative language (at a comparative scale), and see that the performance of both the global and local models lie in this range, so our results are consistent with this hypothesis.

Figure 4 shows that projects separate from the global standard to varying degrees in time, and sometimes from each other. Recall that global model entropy is static; thus, the plot measures how local model performance changes in time *w.r.t.* the static global model. The idea is that if the local model performs better than the global model (lower entropy), the difference (global - local entropy) is larger, leading to a larger y-axis value. As shown, after an initial (slightly chaotic) period, projects generally settle within a region where the local model outperforms the global model, with varying degree, depending on project. This indicates that projects do diverge from the global norm and towards a local norm; *i.e.*, posts tend to conform to the local norm.

Research Answer 1: *We find that we can construct a language model representative of the norm for GitHub overall. We find that perceived project-specific language complexity exists, and has significantly lower complexity than the GitHub standard. Projects diverge from the global standard to varying degrees, and from each other.*

RQ 2a: Is there migration in perceived language complexity, over time, toward the project norm?

Figure 6 contains two plots. Figure 6a contains linear model fits for the separation between global and local model entropies per post in time, for each person, analogous to Figure 4. If the slope of the fit is positive, the separation between the global and local model for each person becomes larger in time; the opposite is true if the fit is negative. Figure 6b is a histogram of slopes following the logic in Figure 6a for people with at

[11] The Brown corpus consists of a mixture of news articles, letters, books, *etc.* from different genres. The Gutenberg corpus consists of books.

least 2 posts. There is a large concentration of users around a slope of 0, indicating that most people do not become further separated by the global and local models in time. Note that a 0 slope does not mean that local norm conformity does not exist, just that the separation may not increase with time. It could be that these individuals are already at the local project norm, and thus have reached a saturation in their ability to separate themselves from the global norm. It could also be that once people hit this saturation point, they do not further conform to the local norm. This is supported by evidence from Posnett *et al.* that users do not become better at asking questions over time on Stack Overflow [46]. It may also be that the project is dominated by "tourists", *e.g.*, those that briefly "visit" the project, post an issue and participate in a single discussion, then leave. These individuals would then not further conform to the local norm, as they are not active participants. To investigate if there is conformance to the local norm for each person, we build models where we control for confounds that may otherwise affect interpretation.

RQ 2b: Do users (popular, experienced, or otherwise) conform to their associated project language? And does perceived language complexity influence popularity?

Table II contains our model describing poster mean entropy (local and global). As this model has user-level observations, the dependent variables are mean entropy across all posts by the given user. For each variable with the description "*as of last post*", we mean the last post by the user under observation, not the last post overall. Poster's GitHub age is the time from the creation of their GitHub account to the time of their last observed post; poster's project age is the time from the first authoring of a commit to the time of their last observed post.

Our affectors of interest are popularity (measured by indegree in the @mention network) and measures of age (poster's GitHub age and poster's project age). We argue that popularity is best measured by indegree as an in-link in the @mention network indicates that someone has mentioned the poster directly in context of the discussion, serving as a measure of how many others are aware of the @mentioned user in the project; thus a proxy for popularity.

For the local model, we see significant negative effects of our age measures. This indicates that older (GitHub age) and more experienced (project age) users have posts with lower local perceived language complexity, indicating conformity to the local norm. We see no significant effect of popularity or code contributions for the local model.

For the global model, we see a similar effect of age as for the local model. However, in the global model, popularity and code contributions are significant as well. A more popular poster has lower global perceived language complexity, but a higher code contributor has a higher global perceived language complexity. The coefficient for code contributions can be explained by the fact that code contributors are likely in-tune with the technical details of a project, and thus discuss them more frequently than others. As technical terms are project-specific, we expect that someone who uses them more has a higher global language complexity, indicating divergence from the global norm. The negative effect of popularity on global language complexity is initially somewhat puzzling - why would a popular person within a project use language fitting to the global norm? When analyzed in tandem with our popularity models, we can investigate this finding.

Table III contains our model for describing poster popularity, which also has user-level observations. We see a positive coefficient for both age metrics, indicating that older and experienced users generally have higher popularity. However, the effects of poster's mean entropy have differing signs: positive for the local case, and negative for the global case. In other words, being more like the global norm (decreased global entropy) leads to higher within-project popularity, and being more like the local norm (decreased local entropy) leads to decreased within-project popularity.

There are a variety of social and technical mechanisms that could explain this. In general, people value outside opinions; prior work has shown that diversity (geographic, cultural, *etc.*) is important in OSS [47], [48]. The explanation could lie in the existence of cross-project correlated bugs [49], where people discuss issues with members of required library projects. Perhaps those with high indegree are most called-upon by people in the out-group (*e.g.*, users within the SC but not the CoP), and thus express themselves using language that out-group individuals would better understand, *i.e.*, the global language. The precise mechanism behind this phenomenon could be the focus of future work.

Research Answer 2: *We find evidence of migration towards a perceived project-specific language complexity norm in time. We find that those with higher GitHub and project age conform to both the local and global norms, while popular users conform more to the global norm. We find that conformity to the local norm leads to decreased popularity, while conformity to the global norm leads to increased popularity.*

RQ 3: What is the relationship, if any, between perceived language complexity and issue resolution latency?

Table IV is our model for describing a software engineering outcome: issue resolution latency. This model has issue-level observations; we look at the first post's entropy and user-level attributes of the poster, along with summary metrics for the rest of the discussion, *e.g.*, mean issue entropy - measured across the entire issue discussion. This choice was made as the first post in an issue generally explains the issue in detail and sets the stage for the following discussion. When "poster" is in the coefficient name, we are referring to the first post (*i.e.*, opening post) in the issue thread. For indegree and code contribution variables, we use the calculated value at the time of posting. The "ordered time of first post" variable is an ordinal time variable that essentially counts the total number of posts across all issues preceding the observed in the project; *e.g.*, if the first post of a given issue is the 100th post overall, the value of this variable will be 100. Age measures are defined by the relevant starting time (account creation or first authoring of a commit) to the time of the post under observation.

The coefficient for local entropy of the first post is positive; those posts which do not conform to the local norm experience $e^{0.023} \approx 2.3\%$ longer response time (in hours) for each bit of increased entropy. We see something similar for the mean entropy of the discussion as a whole: each bit of increased entropy for the local model leads to a $e^{0.116} \approx 12.2\%$ longer response time. The effect for the first post is small in percentage, but as resolution latencies increase, the raw effect can be noticeable. The effect for mean discussion entropy is more palpable; if an issue would otherwise take 8 hours to resolve, an increase to it of 12.2% may take longer than a working day, reducing its usefulness, as responsiveness to important issues (*e.g.*, bugs) is vital for software success. Thus, if one can influence an issue discussion to be more towards the local norm, there can be software related benefits.

> **Research Answer 3**: *Conformity towards local language norm can reduce issue resolution times. Increase in first post entropy leads to* 2.3% *increase in resolution time, while increase in mean entropy of posts across the discussion leads to* 12.2% *increase in resolution time.*

VI. Implication for Software Practice

Here we distill some practice-related conclusions from our studies. Some mix of technical and social conversation may be expected in each conversation in GitHub. From a community perspective, developers may take less time to acculturate if they know ahead of time that such a process is expected, and may in fact be imminent. This result, in conjunction with prior results on work/talk culture in OSS communities [21], points in the direction that strong incompatibility of an individual and a project culture may lead to a failure to acculturate, and result in subsequent departure from the project. Developing explicit acculturation plans, beyond just "learning the ropes" may aid in early discovery of incompatibilities with the project and also prevent attrition when the learning curve is too steep.

We found that smaller perceived language complexity in discussions, *w.r.t.* the local norm, is most helpful to issue resolution latency. So, an abstract recommendation is to have developers use language in issue discussion as close as possible to the project-specific language. There are several ways in which this can be made more specific and even actionable.

Newcomers to a project should be patient and learn the language and culture before actively participating in project issue discussions. This may not be a desirable solution, especially in the cases when developer labor is needed to solve an issue, so the next solution may be more appropriate.

Developers should try and use language consistently that is close in complexity to the project norm. To aid in this we provide a prototype example feedback tool, called *Normative*,[12] which works with ~50 projects for now. When a project is selected, and a developer types a paragraph in the tool's open field, Normative calculates the perceived language complexity of that text to the specified project. Then, the developer can change that text at will and observe how the distance changes. As per feedback learning theory [50], repeated trials in this tool can result in learning to write closer to the norm. We will enhance this tool in the near future with examples from related posts that are similar to the input text.

VII. Conclusion

In this work, we presented an initial look into perceived language complexity from an analysis of communication traces (issue discussions) on GitHub, guided by sociolinguistic theory. We constructed language models representative of a "global" GitHub language, and compared these models to nested "local" project-specific language models, and saw that local language norms exist, and differ by project. We found that users gravitate towards perceived project-specific language complexity norms, as expected by theory, and that popular users are interestingly better represented by the global norm than the related local norm. Also, lack of conformity to the local norm increases issue resolution times, by a small amount. To our knowledge, this is the first work that examines GitHub language complexity and conformity towards project language norms, and their effects on software engineering outcomes. We hope followup work along these lines will understand better how OSS community language style and project-specific quirks affect outcomes important for OSS success.

Threats to Validity: Some model R^2 may be considered low. As discussed in Sect. IV-D, we favored better model diagnostics over higher R^2, so as to not overfit. We did fit models with R^2 between 60% and 70% , but these did not pass diagnostics tests and thus are not valid for inference. We note that lower goodness of fit does not negate the individual effects of the independent variables.

We acknowledge that the effect size of some of our variables are low; *e.g.*, 2.3% increased resolution time for each additional bit of entropy for the first post. Nevertheless, the raw values can be large if latency is high. They also need to be taken in the larger context where the full discussion, if more complex, can add 12.2% to the resolution time for each additional bit of mean entropy, and the first post may set the stage for similar posts.

As with any model, we have the threat of missing confounds. We attempted to control for many aspects that could affect our outcomes, and performed a best-effort to gather as much data as we reasonably could to use in our models.

Our measure of popularity using @mention networks has no precedent in prior work; others have looked into the social network of @mentions, but it is not known whether this measure of popularity is comprehensive. We chose it due to its natural occurence in issue discussions - our focus here.

Our work is an initial foray into the topic of differing language use in GitHub. Mixed method approaches, including qualitative studies, *e.g.*, regarding language use in studied projects, would strengthen future studies on his topic.

Acknowledgments

We thank members of the UC Davis DECAL group for comments and advice regarding this work.

[12] https://github.com/normative-team/normative

References

[1] S. K. Deckert and C. H. Vickers, *An introduction to sociolinguistics: Society and identity*. A&C Black, 2011.

[2] D. Hymes, “Two types of linguistic relativity,” in *Sociolinguistics: proceedings of the UCLA Sociolinguistics Conference*, 1964, pp. 114–67.

[3] K. Varantola, “Special language and general language: Linguistic and didactic aspects,” *Unesco Alsed-LSP Newsletter (1977-2000)*, vol. 9, no. 2, 1986.

[4] J. Lave and E. Wenger, *Situated learning: Legitimate peripheral participation*. Cambridge university press, 1991.

[5] E. Wenger, *Communities of practice: Learning, meaning, and identity*. Cambridge university press, 1998.

[6] E. Wenger, R. A. McDermott, and W. Snyder, *Cultivating communities of practice: A guide to managing knowledge*. Harvard Business Press, 2002.

[7] J. Holmes and M. Meyerhoff, “The community of practice: Theories and methodologies in language and gender research,” *Language in society*, vol. 28, no. 02, pp. 173–183, 1999.

[8] L. Dubé, A. Bourhis, and R. Jacob, “The impact of structuring characteristics on the launching of virtual communities of practice,” *Journal of Organizational Change Management*, vol. 18, no. 2, pp. 145–166, 2005.

[9] J. Kleinnijenhuis, B. van den Hooff, S. Utz, I. Vermeulen, and M. Huysman, “Social influence in networks of practice: An analysis of organizational communication content,” *Communication Research*, vol. 38, no. 5, pp. 587–612, 2011.

[10] C. Llamas, L. Mullany, and P. Stockwell, *The Routledge companion to sociolinguistics*. Routledge, 2006.

[11] P. Eckert, “Communities of practice,” *Encyclopedia of language and linguistics*, vol. 2, no. 2006, pp. 683–685, 2006.

[12] J. S. Justeson and S. M. Katz, “Technical terminology: some linguistic properties and an algorithm for identification in text,” *Natural language engineering*, vol. 1, no. 01, pp. 9–27, 1995.

[13] W. R. Fisher, “Human communication as narration: Toward a philosophy of reason, value, and action,” 1989.

[14] G. A. Hauser, L. C. Hawes, G. L. Wilson, G. Cheney, P. K. Tompkins, C. R. Burgchardt, C. J. Stewart, E. C. Clark, J. M. Hogan, F. J. Boster, G. M. Phillips, R. T. Craig, S. B. Shimanoff, C. Oravec, J. R. Bennett, E. Smokewood, C. L. Bartow, J. Blankenship, M. P. Graves, R. J. Connors, C. Kramarae, G. Berquist, R. M. Ogles, S. R. Brydon, S. R. Hankins, W. M. Purcell, V. O’Donnell, B. K. Duffy, S. H. Browne, M. Weiler, M. Cooper, and W. R. Fisher, “Book reviews,” *Quarterly Journal of Speech*, vol. 74, no. 3, pp. 347–400, 1988.

[15] D. Birman, *Acculturation and human diversity in a multicultural society.* Jossey-Bass, 1994.

[16] R. H. Teske and B. H. Nelson, “Acculturation and assimilation: A clarification,” *American Ethnologist*, vol. 1, no. 2, pp. 351–367, 1974.

[17] P. G. Zimbardo, “Involvement and communication discrepancy as determinants of opinion conformity.” *The Journal of Abnormal and Social Psychology*, vol. 60, no. 1, p. 86, 1960.

[18] E. P. Dozier, “Two examples of linguistic acculturation: The yaqui of sonora and arizona and the tewa of new mexico,” *Language*, vol. 32, no. 1, pp. 146–157, 1956.

[19] M. Lea, D. Barton, and K. Tusting, “Communities of practice in higher education,” *Beyond communities of practice: Language, power and social context*, pp. 180–197, 2005.

[20] Q. Xuan, M. Gharehyazie, P. T. Devanbu, and V. Filkov, “Measuring the effect of social communications on individual working rhythms: A case study of open source software,” in *Social Informatics (SocialInformatics), 2012 International Conference on*. IEEE, 2012, pp. 78–85.

[21] Q. Xuan, P. Devanbu, and V. Filkov, “Converging work-talk patterns in online task-oriented communities,” *PloS one*, vol. 11, no. 5, p. e0154324, 2016.

[22] A. N. Kolmogorov, “Three approaches to the quantitative definition ofinformation’,” *Problems of information transmission*, vol. 1, no. 1, pp. 1–7, 1965.

[23] T. M. Cover, P. Gacs, and R. M. Gray, “Kolmogorov’s contributions to information theory and algorithmic complexity,” *The annals of probability*, vol. 17, no. 3, pp. 840–865, 1989.

[24] T. M. Cover and J. A. Thomas, *Elements of information theory*. John Wiley & Sons, 2012.

[25] I. Kontoyiannis, *The complexity and entropy of literary styles*. Department of Statistics, Stanford University, 1997.

[26] P. Juola and R. H. Baayen, “A controlled-corpus experiment in authorship identification by cross-entropy,” *Literary and Linguistic Computing*, vol. 20, no. Suppl, pp. 59–67, 2005.

[27] H. Kwon, H. T. Kwon, and W. C. Yoon, “An information-theoretic evaluation of narrative complexity for interactive writing support,” *Expert Systems with Applications*, vol. 53, pp. 219–230, 2016.

[28] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb, “Social coding in github: transparency and collaboration in an open software repository,” in *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*. ACM, 2012, pp. 1277–1286.

[29] J. Tsay, L. Dabbish, and J. Herbsleb, “Influence of social and technical factors for evaluating contribution in github,” in *Proceedings of the 36th international conference on Software engineering*. ACM, 2014, pp. 356–366.

[30] K. Blincoe, J. Sheoran, S. Goggins, E. Petakovic, and D. Damian, “Understanding the popular users: Following, affiliation influence and leadership on github,” *Information and Software Technology*, vol. 70, pp. 30–39, 2016.

[31] Y. Yu, H. Wang, V. Filkov, P. Devanbu, and B. Vasilescu, “Wait for it: Determinants of pull request evaluation latency on github,” in *Mining Software Repositories (MSR), 2015 IEEE/ACM 12th Working Conference on*. IEEE, 2015, pp. 367–371.

[32] Y. Zhang, H. Wang, G. Yin, T. Wang, and Y. Yu, “Exploring the use of@-mention to assist software development in github,” in *Proceedings of the 7th Asia-Pacific Symposium on Internetware*. ACM, 2015, pp. 83–92.

[33] V. J. Hellendoorn and P. Devanbu, “Are deep neural networks the best choice for modeling source code?” in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2017. New York, NY, USA: ACM, 2017, pp. 763–773. [Online]. Available: http://doi.acm.org/10.1145/3106237.3106290

[34] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, “A neural probabilistic language model,” *Journal of machine learning research*, vol. 3, no. Feb, pp. 1137–1155, 2003.

[35] S. F. Chen and J. Goodman, “An empirical study of smoothing techniques for language modeling,” in *Proceedings of the 34th annual meeting on Association for Computational Linguistics*. Association for Computational Linguistics, 1996, pp. 310–318.

[36] D. A. Belsley, E. Kuh, and R. E. Welsch, *Regression diagnostics: Identifying influential data and sources of collinearity*. John Wiley & Sons, 2005, vol. 571.

[37] F. L. Schmidt and J. E. Hunter, *Methods of meta-analysis: Correcting error and bias in research findings*. Sage publications, 2014.

[38] M. Hu, “What does it mean to have a low r-squared? a warning about misleading interpretation,” http://humanvarieties.org/2014/03/31/what-does-it-mean-to-have-a-low-r-squared-a-warning-about-misleading-interpretation/#more-3185. Human Varieties, 2014.

[39] P. Birnbaum, “On correlation, r, and r-squared,” http://blog.philbirnbaum.com/2006/08/on-correlation-r-and-r-squared.html. Sabermetric Research, 2006.

[40] P. Birnbaum, “r-squared abuse,” http://blog.philbirnbaum.com/2007/10/r-squared-abuse.html. Sabermetric Research, 2007.

[41] J. Cohen, *Applied multiple regression/correlation analysis for the behavioral sciences*. Lawrence Erlbaum, 2003.

[42] J. Cohen, “Statistical power analysis for the behavioral sciences (revised ed.),” 1977.

[43] S. Hasan and H. Ney, “Clustered language models based on regular expressions for smt,” in *Proc. of the 10th Annual Conf. of the European Association for Machine Translation (EAMT)*. Citeseer, 2005.

[44] A. Hindle, E. T. Barr, Z. Su, M. Gabel, and P. Devanbu, “On the naturalness of software,” in *Software Engineering (ICSE), 2012 34th International Conference on*. IEEE, 2012, pp. 837–847.

[45] Z. Tu, Z. Su, and P. Devanbu, “On the localness of software,” in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 2014, pp. 269–280.

[46] D. Posnett, E. Warburg, P. Devanbu, and V. Filkov, “Mining stack exchange: Expertise is evident from initial contributions,” in *Social Informatics (SocialInformatics), 2012 International Conference on*. IEEE, 2012, pp. 199–204.

[47] B. Vasilescu, A. Capiluppi, and A. Serebrenik, “Gender, representation and online participation: A quantitative study of stackoverflow,” in *Social Informatics (SocialInformatics), 2012 International Conference on*. IEEE, 2012, pp. 332–338.

[48] B. Vasilescu, D. Posnett, B. Ray, M. G. van den Brand, A. Serebrenik, P. Devanbu, and V. Filkov, "Gender and tenure diversity in github teams," in *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. ACM, 2015, pp. 3789–3798.

[49] W. Ma, L. Chen, X. Zhang, Y. Zhou, and B. Xu, "How do developers fix cross-project correlated bugs?" To be presented at ICSE, 2017.

[50] H. Pashler, N. J. Cepeda, J. T. Wixted, and D. Rohrer, "When does feedback facilitate learning of words?" *Journal of Experimental Psychology: Learning, Memory, and Cognition*, vol. 31, no. 1, p. 3, 2005.