Linguistic Change in Open Source Software

A place holder for authors' names
Authors' affiliation
Authors' address
Authors' emails

***Abstract*—In this paper, we seek to advance the state-of-the-art in code evolution analysis research and practice by statistically analyzing, interpreting, and formally describing the evolution of code lexicon in Open Source Software (OSS). The underlying hypothesis is that, similar to natural language, code lexicon falls under the remit of evolutionary principles. Therefore, adapting theories and statistical models of natural language evolution to code is expected to provide unique insights into software evolution. Our analysis in this paper is conducted using 2,000 OSS systems sampled from a broad range of application domains. Our results show that a) OSS projects exhibit a significant shift in their linguistic identity over time, b) different syntactic structures of code lexicon evolve differently, c) different factors of OSS development and different maintenance activities impact code lexicon differently. These insights lay out a preliminary foundation for modeling the linguistic history of OSS projects. In the long run, this foundation will be utilized to provide support for basic software maintenance and program comprehension activities, and gain new theoretical insights into the complex interplay between linguistic change and various system and human aspects of OSS development.**

I. Introduction

Open Source Software (OSS) is a unique phenomenon in Software Engineering. Unlike traditional commercial software, the idea of OSS is expressed through volunteer participation of programmers who work together to write code, make changes, debug, and maintain the product. Over the past 20 years, OSS has become a significant factor in driving the growth of software industry, with major technological companies, such as Google, Microsoft, and IBM, turning to open source as a more reliable, cost-effective, and secure development paradigm [1].

The tremendous success of the OSS movement over the past decade has been boosted by the emergence of online version control systems, such as *GitHub* and *SourceForge*. These platforms support an integrated set of technical and social features that enabled software enthusiasts to establish world-wide social networks of developers at unprecedented scales. This form of open and decentralized development has forced OSS projects to adopt more responsive development strategies (e.g., continuous integration and deployment) in order to control for quality and meet market pressures [2]. This rapid pace of change imposes a substantial evolutionary pressure on OSS projects' source code. Observing, understanding, and modeling the dynamics of this evolution enable developers to diagnose and reverse the symptoms of code aging, make informed design and maintenance decisions, and ensure a sustainable and stable delivery process [3]. However, current version control systems provide a single lens on software history through commits. Commits record changes per file. Analysis of online software repositories have reported that commits and issue tracking systems often hold incomplete or incorrect data, where a large percentage of change information is rarely documented or poorly reported [4].

To overcome these limitations, in this paper, we introduce a new window on software history. In particular, we analyze software evolution through the lens of developers' language, also known as the code lexicon [5]. The vocabulary of this language, embedded in code identifier names (e.g., method, variable, class, and package names, etc.) and internal code comments, captures developers' understanding of their system and its application domain at the most primitive level [6]. Our main hypothesis is that words in a software system's vocabulary exhibit evolutionary patterns similar to natural language words, influenced by various human (developers joining and departing) and system (maintenance activities) factors of OSS development. This hypothesis is further supported by the overwhelming evidence of the natural attributes of code lexicon and its evolutionary characteristics [7], [8].

Our theoretical foundation in this paper builds upon several modern theories of natural language evolution [9], [10]. These theories are proposed to understand, explain, and predict linguistic change as a parameter of cultural and social anthropology [9], [10]. To conduct our analysis, we collect and analyze a large-scale dataset of OSS projects, their revisions, and metadata. Our contribution lies in testing several assumptions related to the complex interplay between linguistic change and various system and human aspects of OSS development. Our objective is to gain a new perspective into the evolution of code lexicon in OSS projects as well as provide support for basic software maintenance and comprehension activities.

The reminder of this paper is organized as follows. Section II motivates our research. Section III describes our research questions, data collection, and preliminary analysis. Section IV discusses the potential implications of our work. Finally, Section V concludes the paper.

II. Background and Motivation

Linguistic analysis of code has revealed that developers' vocabulary evolves over time, reflecting various types of changes that took place between the different revisions of the system [11], [5]. Such changes might range from a simple variable rename, to more complex changes, such as fixing a bug or adding a feature. Antoniol et al. [5], provided a preliminary evidence that structural and lexical stabilities of

software systems do not have the same distribution. A follow up study by Abebe et al. [11] found that the vocabulary used by programmers is subject to an evolutionary pressure similar to that affecting code structure. Both studies acknowledge the fact that the evolution of code lexicon does not follow a trivial pattern and more research is needed to fully understand it. Furthermore, several researchers have exploited language modeling techniques (e.g., n-grams [7]) and text power laws (e.g., Zipf's and Heap's laws), to explore statistical regularities in code [12], [8]. The results have largely confirmed previous speculations about the naturalness of code lexicon and provided a significant evidence on the presence of text power laws in code. However, these algorithms are often applied on static snapshots of software systems, ignoring the inherently dynamic nature of code. Furthermore, exiting work often ignores the rule of human factors in linguistic change. However, developer teams in OSS are inherently diverse, mixing and uniting people from different countries and different development backgrounds. Whenever a new developer joins the team, the project's vocabulary might shift based on their linguistic preferences and previous syntactic habits, making OSS projects more prone to linguistic change than other, more structured, development environments.

Motivated by **a)** the tremendous growth, unprecedented success, and wide-spread of OSS products in the software market **b)** the overwhelming evidence of the natural attributes of code lexicon and its evolutionary characteristics, **c)** the significant information value of the project vocabulary for software developers, and **d)** the limitations of existing code evolution analysis techniques and tools, in this paper, we propose to study code evolution in OSS projects through the lens of developers language. Our main hypothesis is that, extracting, analyzing, and statistically modeling linguistic change during OSS projects evolution will provide a new window on software history, revealing unique aspects of code evolution that are typically overlooked by existing methods often available in modern version control systems, such as observing code file and code line changes.

III. Preliminary Analysis

In OSS development, the frequent maintenance actions performed on the system as well as the highly dynamic nature of developer teams, apply immense evolutionary pressures on the usage and survival capacity of code lexicon words. To understand and quantify the magnitude of this change, we formulate the following research questions:

- ***RQ1***: *Do OSS projects experience linguistic change over time?*
- ***RQ2***: *What OSS development factors impact linguistic change?*
- ***RQ3***: *Do different syntactic forms of code lexicon evolve differently?*

These questions are intended to provide a fundamental understanding of the dynamics of linguistic change in OSS projects. Answering these questions will help us to gain new theoretical insights into code evolution and formulate a research vision on how such insights can be utilized to enable a more sustainable OSS development process.

A. Data Collection

To conduct our analysis, a large number of OSS projects, their revisions, and metadata need to be collected from platforms such as *GitHub* and *SourceForge*. In our analysis, we focus on GitHub as our main data collection platform. To select the projects to include in our dataset, we rely on GitHub's starring system. Numerous studies reported that the number of stars tends to correlate with project success, or the number of forks, pull requests, commits, and comments a project receives [13]. To collect our data, we used the GitHub API to download the top starred Java, C#, Python, and JavaScript projects (500 x 4) along with their corresponding public releases. Descriptive statistics of our dataset are provided in Table I.

To extract source code lexicon from our projects, we used regular expressions following a procedure first introduced by Khatiwada et al. [14]. Code lexicon consists of all words used in the source code except for the programming language specific keywords. Such words are available in comments, class and variable declarations, class and variable usage, method declarations, parameters, generic types, and other language-specific constructs, such as annotations in Java or properties in C#. After identifiers are extracted, we further split any compound words into their constituent words based on camel-casing (e.g., `userID` is split into `User` and `ID`) or any special characters (e.g. underscore) typically used in code naming conventions (e.g., `file_type` is split into `file` and `type`). After atomic words of code identifiers are extracted, stemming (Porter stemmer) is applied to reduce words to their morphological roots.

B. Quantifying Linguistic Change in OSS projects

In the first phase of our analysis, we are interested in the magnitude of linguistic shift in OSS projects. Based on Petersen et al.'s word-frequency model [10], the linguistic change rate ($\Delta\lambda_{i,j}$) between two releases r_i and r_j of the system can be calculated as the number of different words (words birth and death) between the two releases divided by the number of unique words in both releases. The average λ_s of a system s which has n releases can be calculated as the average $\lambda_{i,j}$ between each two consecutive releases in the time series of the system:

$$\Delta\lambda_{i,j} = 1 - \frac{|r_i \cap r_j|}{|r_i \cup r_j|}, \quad \lambda_s(t) = \frac{1}{n}\sum_{i=1}^{n-1}(\Delta\lambda_{i,i+1}) \qquad (1)$$

To answer ***RQ1***, we calculated the linguistic difference between the first and last release ($\lambda_{0,n}$) of each system in our dataset. The results are presented in Fig. 1, which shows the frequency distribution of linguistic change in code and comment overlapped, sorted in an ascending order. The distribution shows that, between the first and most recent releases, around 50% of the projects experienced more than 40% and 50% of linguistic shift in their code and comments respectively.

TABLE I: Descriptive statistics of our dataset of OSS projects.

Language	Total # of Projects	Total # of Releases	Average # of Releases	Total kLOC	Average kLOC
Java	442	9,947	20.60	728,504.91	1,508.29
C#	480	13,482	26.23	1,330,252,20	2,588.04
Python	472	11,818	22.51	835,544,98	1,591,51
JavaScript	481	19,861	39.56	1,162,249,94	2,315,24

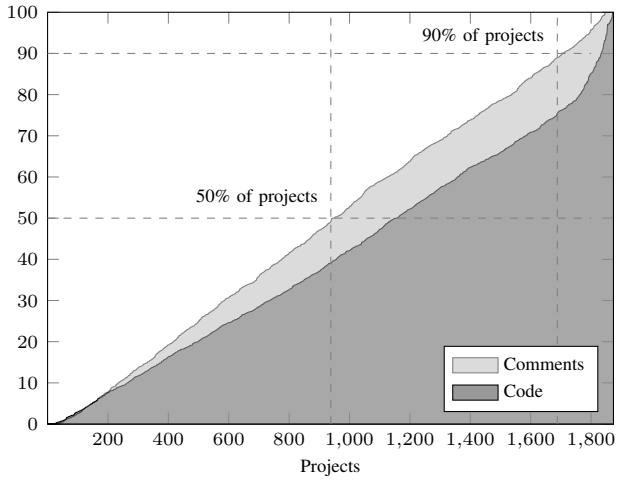

Fig. 1: The frequency distribution of $\lambda_{0,n}$ for the GitHub projects in our dataset

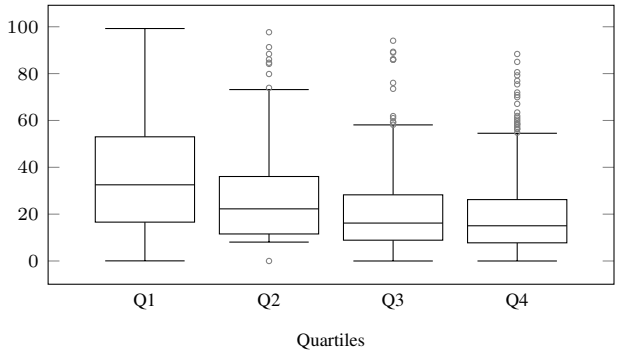

Fig. 2: Average linguistic change rate at different release quartiles

Around 5% of the projects have witnessed more than 90% of change in their linguistic identity (code and comments). The results also show that linguistic change in comments was more prominent than that of the code. To estimate the magnitude of linguistic change at different stages of the project's lifecycle, we measured the average shift in the project's language at different development quartiles. Fig. 2 shows that changes tend to be more severe at the early quartiles. However, projects lean toward stability as time goes by, nonetheless, the change is still significant (12-15%). In summary, we can conclude that majority of OSS projects exhibit significant shift in their linguistic identity over time.

C. OSS Factors vs. Linguistic Change

Under this phase of our analysis, we investigate how linguistic change correlates with three main measures of OSS projects activity (***RQ₂***). These measures, or indicators of popularity [15], can be described as follows:

- **Number of contributors**: OSS projects unite developers (contributors) from around the globe. Therefore, we expect higher linguistic change levels to be associated with higher number of contributors.
- **Number of releases**: a release is a distinct version of the system that is substantially different from the previous version. Our expectation is that more releases indicate more change, thus higher levels of linguistic change.
- **Number of commits**: the number of commits is often used as a proxy of OSS team's productivity [15]. Consequently, projects with higher number of commits are expected to exhibit larger levels of lexicon shift.

To test if these relationships are statistically significant, we conducted an Ordinary Least Squares (OLS) Regression analysis for each programming language. The indicators were log-transformed to meet the normality assumption. The results of the analysis are presented in Table II. In general, the number of contributors seem to have an impact on the linguistic change of code for Java, C#, and JavaScript. It also showed statistical significance with comments' linguistic change for C# and JavaScript. However, it failed to reach the level of statistical significance for Python, both for code and comments. The number of releases consistently predicted linguistic change for code and comments across all programming languages, with high statistical significance. In addition, this variable showed a relatively good exploratory power of the variance in the dependent variables by achieving decent R^2 values. Overall, the number of releases turned out to be a much stronger predictor of the linguistic change than all other indicators. The number of commits consistently showed significance across all languages. A smaller R^2 values suggest that number of commits is not enough to explain the variation in the dependent variables. Nevertheless, the relationship is strong and positive. In summary, linguistic change can be significantly predicted by the number of published releases and commits. The number of contributors, however, tends to show mixed results.

D. Lexicon Words Fitness

To examine the impact of the syntactic structure of code lexicon words on their fitness (***RQ₃***), we analyzed the survival rates of misspelled words and short forms (single letter identifiers, abbreviations, and acronyms) in comparison to

TABLE II: Regression analysis of the impact of different factors of OSS development on the linguistic change of code and comments (cmnt) in different programming languages.

		Java		C#		Python		JavaScript	
		Code	**Cmnt**	**Code**	**Cmnt**	**Code**	**Cmnt**	**Code**	**Cmnt**
Contributors	B	$6.91^{\dagger}$	1.20	$6.25^{\dagger}$	6.15^{*}	3.11	0.11	$14.52^{\ddagger}$	$11.81^{\ddagger}$
	R^2	0.03	0.00	0.02	0.01	0.00	0.00	0.07	0.04
Releases	B	$27.99^{\ddagger}$	$27.80^{\ddagger}$	$26.65^{\ddagger}$	$31.65^{\ddagger}$	$24.24^{\ddagger}$	$25.47^{\ddagger}$	$27.04^{\ddagger}$	$26.06^{\ddagger}$
	R^2	0.28	0.18	0.23	0.24	0.18	0.15	0.21	0.16
Commits	B	$7.00^{\ddagger}$	3.94^{*}	$6.78^{\ddagger}$	$6.83^{\dagger}$	$5.03^{\dagger}$	3.32	$12.17^{\ddagger}$	$8.65^{\ddagger}$
	R^2	0.05	0.01	0.03	0.03	0.02	0.00	0.07	0.03

*Note: *p<0.5; †$p < 0.01$; ‡$p < 0.001$*

correctly spelled natural words in our 2,000 sample projects. Misspellings were identified using Levenshtein distance and a natural language dictionary. Based on existing literature, it makes sense to assume that natural language words will be more syntactically fit than short forms, while misspellings will be the least fit. Based on Petersen et al. [10], the fitness of a word in a project's vocabulary of size *n* at a specific release (*t*) can be quantified as the number of times that word appears in the release $u_i(t)$ divided by the total number of words in the release $N_u(t)$, such that:

$$f_i(t) = \frac{u_i(t)}{N(t)}, \qquad N(t) = \sum_{i=1}^{n} u_i(t) \tag{2}$$

The word's birth is marked as the first release where $f_i(t) > 0$, and the release of death is the release at which the word disappears $f_i(t) = 0$. The average fitness of a word over its lifetime in a project of n releases, can be calculated as:

$$\langle f_i \rangle = \frac{1}{n} \sum_{j=1}^{n} f_j(t). \tag{3}$$

The results of our lexicon fitness analysis, based on word syntactic type (natural, short, and misspelling) is shown in Fig. 3. In general, the results came out as expected. Natural language words are the fittest, followed by short forms (abbreviations), followed by misspellings. In summary, we can conclude that different syntactic forms (type and length) have different survival capacity, or fitness. In what follows, we discuss the implications of these findings in greater detail.

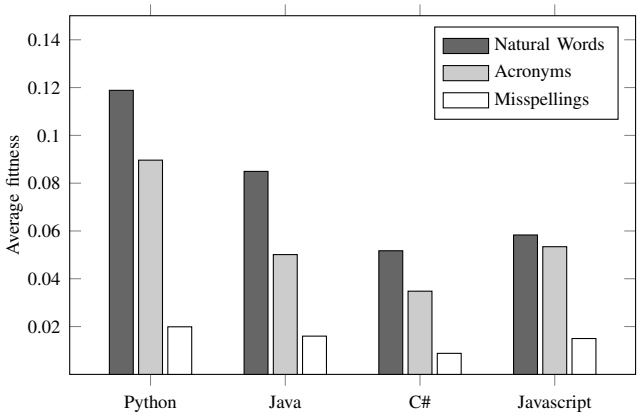

Fig. 3: Syntactic fitness for various word forms

IV. Discussion and Expected Impact

The main goal of our analysis in this paper is to provide a preliminary evidence on the natural attributes of code lexicon evolution. According to Croft's [16], understanding language change provides a basis for understanding the generation and propagation of language structures, thus provides a description of how a language system may emerge and continue to change over time.

In general, our preliminary analysis has revealed that the majority of OSS projects exhibit a significant shift in their linguistic identity over time. This shift seems to be more drastic at the first quartile of releases, and then tends to happen at almost a steady pace. We also notice that this change is far from being a linear function of the number of releases, contributors, and commits. Furthermore, our preliminary results suggest the presence of a syntactic *Darwinian* effect between words, where less fit syntactic forms (i.e., misspellings and short forms) are more prone to go extinct than their natural counterparts. Another interesting observation is that, some words tend to show significant fluctuation in their utility (disappearing and reappearing multiple times). Examining our projects shows that the majority of these words are common misspellings of natural words (e.g., *idetifies*, *exising*). These findings represent the basis for our future work agenda in this domain. Our aim is to provide a fundamental understating of how conventions in OSS projects emerge and become shaped by selection. Specifically, by analyzing massive numbers of OSS projects, their revisions, and metadata, we seek to advance the state-of-the-art in code evolution analysis research and practice along the following directions:

- **Documenting software history**: Developers frequently ask questions about the history of the code to understand the change rationale, track bugs to their origin, or trace features to their older versions [3]. Code lexicon change is typically influenced by maintenance activities (e.g., adding features, fixing bugs, and refactoring). Therefore, the ability to capture and model the change in the

system's vocabulary is expected to reveal the history of events that led to these changes. An underlying assumption is that different software evolution activities have different impacts on the systems vocabulary. This analysis will enable us to quantify OSS projects' memory by documenting their linguistic change, thus provide a unique window on software history.

- **Supporting code change activities**: Analyzing the fitness of system vocabulary can reveal the importance of the different system features and describe the way the system has evolved to its current state. For instance, by mapping words' fitness to the importance of the code they describe, architecturally significant code can be identified and protected. Using such information, any contribution that is directed toward semantically fit words can be subjected to more intensive code reviews. Furthermore, word fitness analysis can be utilized during code reviews to preserve the syntactic fitness of the project and provide project-specific styling and naming restrictions for OSS development [12]. In fact, such restrictions can be further utilized for devising automated methods for faster style checking, especially that recent research has shown that around 25% of code reviews in OSS development contain suggestions about naming [12].
- **Building OSS teams:** it is important to understand how the internal dynamics of developer teams and the linguistic norms of the project jointly evolve. In their work on online communities, Danescu-Niculescu-Mizil et al. [17] reported that members of an online community who fail to adapt to the evolving linguistic norms of the community are projected to leave the community. Assuming that OSS projects offer an underlying social structure that resembles an online community, it would be important to examine the impact of linguistic change on members (developers) of OSS teams. Specifically, identifying the linguistic habits of OSS developers and their level of linguistic adaptation can help to identify optimal linguistic change rates that can sustain participation levels in OSS team. Such information can aid in predicting, early in developers' career, how long they will stay active in the community. According to Schilling et al. [18], the early identification of developers who are likely to remain is an eminent challenge for the management of OSS initiatives. Furthermore, resolving the relations between linguistic change and other aspects of OSS code development will enable us to uncover best practices for fostering and controlling innovation in OSS development teams. For instance, resolving the relations between linguistic stability, code quality, and developer creativity will enable us to provide practical guidelines for reviewing code contributions. Our goal is to define the levels of linguistic change that can be optimal for keeping the system's quality under control, at the same time, do not restrain OSS developers' creativity.

V. Conclusions

In this paper, we proposed to study code evolution in OSS projects from the unique perspective of developer language. Our main hypothesis is that, in distributed development environments, code lexicon is constantly changing in response to immense evolutionary pressure. Capturing, modeling, and interpreting such change can provide a unique window on software change, providing insights into software history that are often missed by existing code evolution analysis methods. The results provided an initial evidence on the magnitude of lexicon change, its natural attributes, and main driving forces. Based on this evidence, we introduced several directions of future work that are aimed at using linguistic change as a factor to identify best practices for managing and sustaining successful OSS projects and vibrant OSS communities.

Acknowledgment

A place holder for acknowledging the sponsor

References

[1] A. Boulanger, "Open-source versus proprietary software: Is one more reliable and secure than the other?" *IBM Systems*, vol. 44, no. 2, pp. 239–248, 2005.

[2] M. Hilton, T. Tunnell, K. Huang, D. Marinov, and D. Dig, "Usage, costs, and benefits of continuous integration in open-source projects," in *ASE*, 2016, pp. 426–437.

[3] M. Codoban, S. Ragavan, D. Dig, and B. Bailey, "Software history under the lens: a study on why and how developers examine it," in *ICSME*, 2015, pp. 1–10.

[4] T. Zimmermann, S. Kim, A. Zeller, and J. Whitehead, "Mining version archives for co-changed lines," in *MSR*, vol. 6, 2006, pp. 72–75.

[5] G. Antoniol, Y. Gueheneuc, E. Merlo, and P. Tonella, "Mining the lexicon used by programmers during sofware evolution," in *ICSM*, 2007, pp. 14–23.

[6] D. Lawrie, C. Morrell, H. Feild, and D. Binkley, "What's in a name? a study of identifiers," in *ICPC*, 2006, pp. 3–12.

[7] A. Hindle, E. Barr, Z. Su, M. Gabel, and P. Devanbu, "On the naturalness of software," in *ICSE*, 2012, pp. 837–847.

[8] D. Pierret and D. Poshyvanyk, "An empirical exploration of regularities in open-source software lexicons," in *ICPC*, 2009, pp. 228–232.

[9] E. Lieberman, J.-B. Michel, J. Jackson, T. Tang, and M. Nowak, "Quantifying the evolutionary dynamics of language," *Nature*, vol. 449, p. 713, 2007.

[10] A. Petersen, J. Tenenbaum, S. Havlin, and E. Stanley, "Statistical laws governing fluctuations in word use from word birth to word death," *Scientific reports*, vol. 2, p. 313, 2012.

[11] S. Abebe, S. Haiduc, A. Marcus, P. Tonella, and G. Antoniol, "Analyzing the evolution of the source code vocabulary," in *CSMR*, 2009, pp. 189–198.

[12] M. Allamanis and C. Sutton, "Mining source code repositories at massive scale using language modeling," in *MSR*, 2013, pp. 207–216.

[13] B. Vasilescu, Y. Yu, H. Wang, P. Devanbu, and V. Filkov, "Quality and productivity outcomes relating to continuous integration in GitHub," in *FSE*, 2015, pp. 805–816.

[14] S. Khatiwada, M. Kelly, and A. Mahmoud, "STAC: A tool for static textual analysis of code," in *ICPC*, 2016, pp. 1–3.

[15] B. Vasilescu, D. Posnett, B. Ray, M. van den Brand, A. Serebrenik, P. Devanbu, and V. Filkov, "Gender and tenure diversity in GitHub teams," in *CHI*, 2015, pp. 3789–3798.

[16] W. Croft, *Evolution: Language Use and the Evolution of Languages*. Springer Berlin Heidelberg, 2013, pp. 93–120.

[17] C. Danescu-Niculescu-Mizil, R. West, D. Jurafsky, J. Leskovec, and C. Potts, "No country for old members: User lifecycle and linguistic change in online communities," in *WWW*, 2013, pp. 307–318.

[18] A. Schilling, S. Laumer, and T. Weitzel, "Who will remain? An evaluation of actual person job and person team fit to predict developer retention in FLOSS projects," in *HICSS*, 2012, pp. 3346–3455.