Diversification vs. Specialization: A Comparative Study of Social Coding in Python and Java on GitHub

As the largest open-source platform for code contributors to develop software toolkits, GitHub has enabled extensive research on social coding. Nevertheless, previous research has not explored whether and how different programming languages lead to distinct collective coding patterns. Our research investigates the linguistic relativity hypothesis for programming languages: that choice of programming language influences the development practices of programmers. By initially building repository representations from source code, readme text, and co-contribution networks, we investigate differences in the development practices of Python and Java communities on GitHub. We conduct sub-studies about the communities' socio-functional mapping, programmers' contribution diversity, and repository popularity. Our findings indicate that Python, a language that emphasizes flexibility and reusability of code, rewards contributors' ability to produce code suited to many different functional needs. In contrast, Java, primarily used to build complete projects for passive end-users, fosters a community of programmers with highly specialized skills. In the Java community, functionally similar repositories exist within a tight-knit social cluster; and this social affiliation is predictive of a repository's popularity. On the other hand, consistent with the Python community's focus on broadly applicable scripting, it is the function, rather than the social context, that relates more to the popularity of a Python repository.

CCS Concepts: • **Software and its engineering** → **Collaboration in software development**; **General programming languages**; • **Information systems** → Social networks.

Additional Key Words and Phrases: GitHub, programming languages, social coding, linguistic relativity

ACM Reference Format:
. 2018. Diversification vs. Specialization: A Comparative Study of Social Coding in Python and Java on GitHub. In . ACM, New York, NY, USA, 9 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

As functional units mediating the communication between humans and machines, software toolkits (i.e., libraries and packages) have become the building blocks of both industrial software development and personal programming projects. Unlike in other industries where workers manufacture products behind closed doors, developers in the software industry often embrace an open-source mindset to jointly develop community software [12]. Through open-source development, technical details like code files can be shared beyond the development team, and external coders who made valuable modifications or wrote new code may be accepted as parties to the development process [12]. This collaborative way of creating software is often referred to as social coding [7, 17, 24].

GitHub, with millions of open-source projects across different programming languages, has become the dominant platform for social coding. Developers there contribute to GitHub repositories and grow their projects by making new commits or merging pull requests from new contributors. GitHub enables further interaction by allowing users to copy code files from a repository by forking, "star" a repository to signify approval, "watch" a repository for new updates, and post issues if they encounter bugs while using the software. The diversity and openness of development activities

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

has prompted research about several aspects of social coding on GitHub, such as collaborative structures and efficiency [3, 24, 26], repository popularity growth [5, 11], and coder expertise and success [6, 8, 17]. Nevertheless, previous studies have not explored the impact of programming language traits on social coding activities.

Although unstudied in the context of programming languages, the notion that the languages people use differentiate their thoughts (known as the Sapir-Whorf or linguistic relativity hypothesis) has been extensively discussed in academic literature from linguistics to the social sciences [14, 15, 20]. Evidence from natural language studies indicates that while languages alone may not constrain cognition, they can influence it [1]. Moreover, many programming language designers, such as Kenneth E. Iverson (creator of APL) and Yukihiro Matsumoto (Ruby), expressed consideration of users' programming ideology in their design [9, 13]. These studies and statements offered us theoretical intuitions to hypothesize that programmers who use different coding languages will possess distinct programming philosophies and behave differently on GitHub. Our goal is to identify and explore this heterogeneity from data stored in repositories produced by programmers of different languages.

We investigate this hypothesis by comparing the Python and Java communities on GitHub, based on their volume and oft-discussed dissimilarity.[1] In this paper, we explore divergences between these two communities as guided by the following questions:

(1) **RQ1 (Socio-functional Mapping)**: Do contributors socially-organize by repository functionality?
(2) **RQ2 (Contribution Diversity)**: Are contributors with diverse experiences likely to be involved in more (or less) popular repositories?
(3) **RQ3 (Popularity)**: What factors are more important for a repository's popularity, social affiliation or functionality?

To investigate these questions, we collect data on repositories' functionality ("*what* they do"), their social context ("*who* created them"), and how popular they are ("*how many* people "star" them"). We use this data to generate dense representations at the repository level based on information drawn from each repository's imported libraries (representing the implementation of a repository's functionality), each repository's Readme text (representing the description of a repository's functionality), and each repository's list of contributors. Each approach generates a unique representation space in which similar repositories are embedded close to one another. Based on these embeddings, we can then explore the differences between the two programming communities.

2 DATA COLLECTION AND PROCESSING

In this section, we describe the data used for our research, as well as the modeling approach to train repository representations.

2.1 Data Collection

We used Python libraries `pygit2`[2] and `PyGithub`[3] to collect GitHub data from repositories downloaded from Github in late 2018. For each repository, we fetched the data from its latest commit

[1]Oracle, the primary developer of Java, describes it as: "a general-purpose, concurrent, strongly typed class-based object-oriented language. It is normally compiled to the bytecode instruction set and binary format defined in the Java Virtual Machine Specification." Python.org describes Python as "powerful... and fast; plays well with others; runs everywhere; is friendly & easy to learn; is Open."
[2]https://www.pygit2.org/
[3]https://github.com/PyGithub/PyGithub

and organized them into four categories, including source code[4], readme documents, contribution records, and metadata[5], where source code and readme documents are relevant to repository functionality [18, 25] and contribution records reflect social context. Our final data set contains 190k Python repositories and 176k Java repositories.

2.2 Learning Representations

Next, we embedded the repositories with representation learning methods on data from the first three categories.

Source Code. For each file of source code in each repository, we began by extracting the abstract syntax tree (AST) of that file. For Python, we used the built-in `ast` module. For Java, we use `javalang`[6]. Using the computed ASTs, we identified the imported libraries in each file, which are highly informative of repositories' functionality. For example, Python deep learning repositories usually import libraries like `tensorflow` and `torch`. To avoid overly-granular library information, we keep only the names of the top-level library in each import statement (e.g., "scipy.stats" becomes "scipy") and aggregate the library names per file. To remove custom libraries designed for individual purposes, we keep only those libraries imported over ten times in the corpus and by at least one repository with five or more stars. We then generated import-based embeddings by extending the Import2Vec method [23] to document-level representations [19]. We generate two distinct representation spaces, for Python and Java, separately. Besides, it was found that Python repositories have considerably more imports than that of Java on average (see Table 1), and we added the number of unique libraries to the metadata of each repository.

Readme Documents. We transformed the readme files into HTML with `pypandoc`[7] and then extracted the content in paragraph and list elements with `beautifulSoup`[8]. Special characters, digits, and email addresses are removed via `gensim`'s preprocessing pipeline. [22] After cleaning, we removed those not in English based on the detection results of `langdetect`[9]. Then we generated readme-based embeddings with the Doc2Vec model [16] and added the length of cleaned Readme text to repositories' metadata.

Contribution Record. We generated a co-contributor network for repositories in each programming language community from their contribution records. The nodes are the repositories not isolated (i.e., having at least one co-contributor with others), and the edges are weighted by the number of users who contributed to both repositories. We then generated contributor-based embeddings with the Node2Vec algorithm [10] and added the repositories' degrees to their metadata. Besides, we discovered that the Python repositories share more contributors than the Java repositories in general (see Table 1).

3 ANALYSIS

In this section, we discuss the methodological details in answering the research questions and the corresponding findings.

[4]We only preserved code scripts in the corresponding language, i.e., .java files for the Java repositories, .py files for the Python repositories.
[5]Original repository metadata includes the number of programming languages, the number of files, the number of contributors, the number of commits, the average number of commits per contributor and the number of stars.
[6]https://github.com/c2nes/javalang
[7]https://github.com/NicklasTegner/pypandoc
[8]https://pypi.org/project/beautifulsoup4/
[9]https://pypi.org/project/langdetect/

language	Python	Java
average number of imported libraries	16	3
average length of cleaned Readme	195	175
average degree of co-contributor network	569	75

Table 1. Key statistics for the Python and Java repositories retrieved from GitHub (values are rounded to integers).

3.1 RQ1: Socio-functional Mapping

RQ1 investigates whether there is an overlap between the social context of a repository and its function. We aim to capture the degree of association between the social and the functional role of a repository by measuring the consistency between the embedding space from functionality features (import-based or readme-based embeddings) and the space from social features (contributor-based embeddings). Boggust et al. [4] described a symmetrical method for measuring consistency between two embedding spaces by computing the Jaccard index of nearest neighbor sets across spaces. However, we found that this approach does not work well in our context due to the large number of repositories analyzed. Instead, we introduce a modified asymmetric method called *Neighbor Distance Comparison*, as follows:

(1) Given two embedding spaces A and B and a neighbor threshold d, select a repository r and find its neighbors N in A whose cosine distance to r is less than the threshold d.
(2) Compute the mean cosine distance: $n \in N$ and r in B as $D_{AB}(d,r) = \frac{1}{|N|}\sum_{n\in N} \text{dist}(n,r)$.
(3) Replicate steps 1 and 2 for all the repositories ($r \in R$) and obtain $D_{AB}(d) = \frac{1}{|R|}\sum_{r\in R} D_{AB}(d,r)$
(4) Replicate step 3 for a sequence of d, and compare the change of d with the change of $D_{AB}(d)$. If A and B are consistent, we expect to observe a correlation between $D_{AB}(d)$ and d: a narrow initial threshold in A should correspond to relative closeness in B, and a wider threshold in A should correspond to relative distance in B.

Using this method, we performed comparisons between the functionality-based representations and the contributor-network-based representations. Figure 1 shows that the consistency between repositories' functionality and contributors' connection exists for the Java community, but not for the Python community, and the pattern is robust regardless of using imported libraries or readme text as the functionality space. Therefore, Java programmers are more organized by repository functionality than Python peers. Moreover, Figures 1(a) and 1(b) reveal that for Python repositories close in the social space (sharing many contributors), their functionality is more different than those in the Java community. This indicates that Python contributors have more tendency to commit to functionally dissimilar repositories than Java contributors. Additionally, Figures 1(c) and 1(d) suggest that in the Python community, a given functionality has a higher chance to be adopted across the social space, indicating fewer barriers to the spread of coding techniques than in the Java community.

3.2 RQ2: Contribution Diversity

Motivated by the findings in 3.1, we aim to determine if the diversity in a contributor's background is more valued by the Python community. We measured the contribution diversity of a contributor c with the average cosine distance between the repositories to which c contributed in a function-based (import-based or readme-based) embedding space:

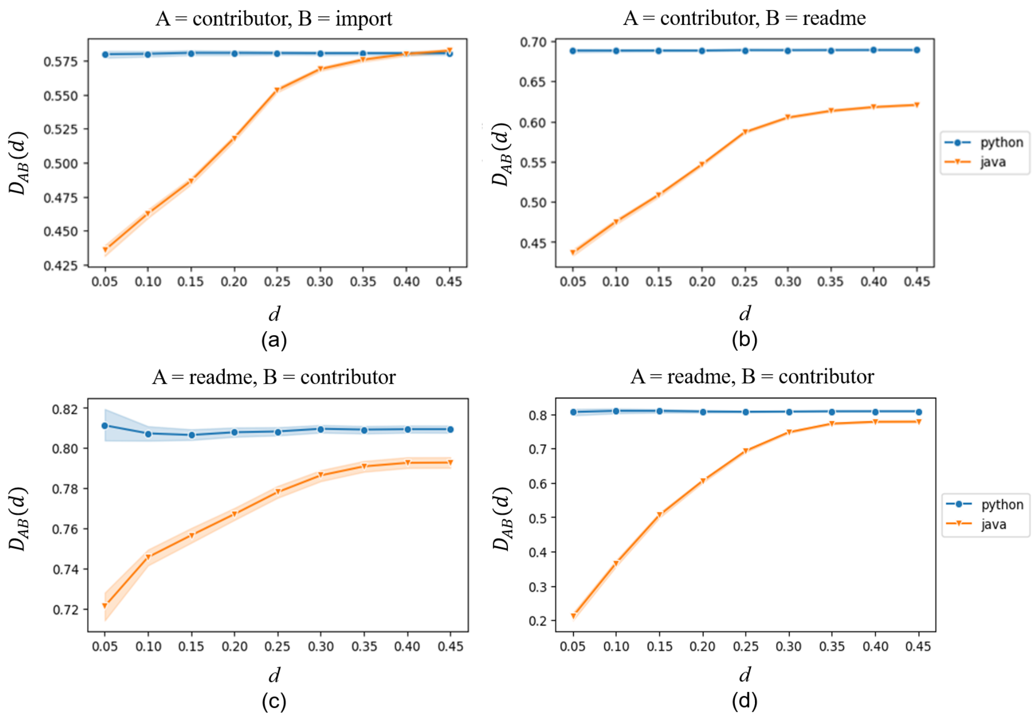

Fig. 1. Neighbor Comparison Results. The neighborhood measurement threshold in space A is recorded on the x-axis, and the corresponding distance score in space B is recorded on the y-axis.

$$\text{Diversity}(c) = \frac{2}{|R_c|(|R_c|-1)} \sum_{i,j|i,j\in R_c, i\neq j} (1 - \frac{i \cdot j}{||i||||j||}),$$

where R_c is the set of function-based embeddings for all repositories contributed to by c. Contributors with only one valid repository were treated as zero-diversity contributors, and we only include meaningful repositories with at least five stars and at least five commits by one contributor.[10]

We then examine the Pearson correlation between contributors' diversity and the average number of stars in the repositories to which they made contributions, which is a measure of the repositories' overall popularity.[11] Additionally, we conditioned on the number of contributed repositories. The result in Figure 2 shows positive correlations for Python contributors, especially when they are actively involved with five or more repositories, and the correlations are stronger than those for Java programmers in terms of both significance and magnitude. More importantly, the discrepancy between the two communities is more evident for contributors committing to larger numbers of repositories, where the correlation coefficient increases for Python but becomes non-significant for Java. These trends are consistent, either using import-based or readme-based embeddings to compute diversity. They also indicate that Python programmers with diverse experience are likely to be engaged with more popular repositories, but such appreciation of diversity is not reflected in the Java community.

[10]The thresholds for the number of stars and commits are tested between 2 and 5, and the correlations are consistent.
[11]Total and median number of stars are also tried, and the correlations are consistent.

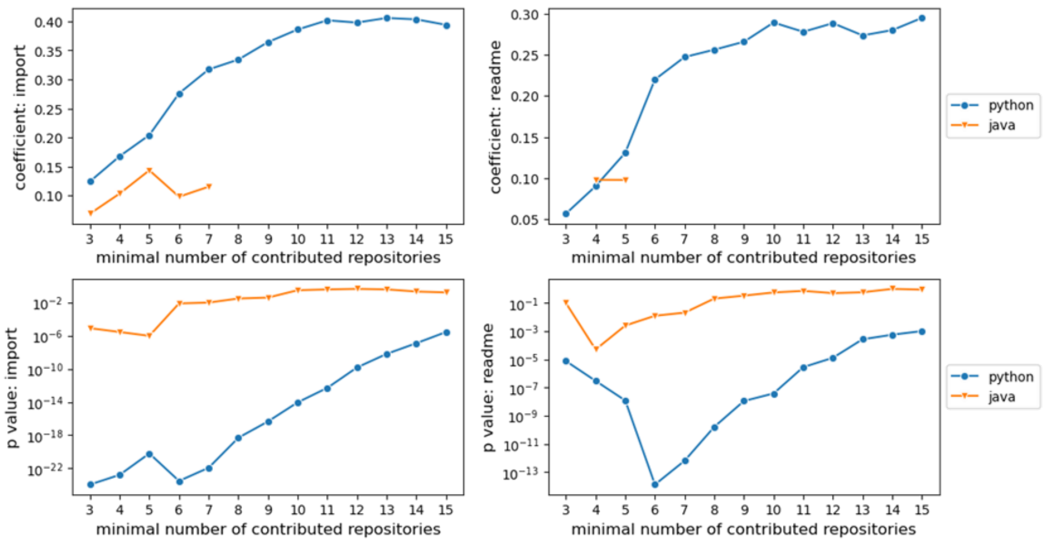

Fig. 2. Correlations between contributing diversity and repositories' average popularity, only statistically significant coefficients (p<0.01) are plotted

3.3 RQ3: Repository Popularity

We performed a regression task with the pooled embeddings to analyze the importance of different features in predicting the popularity of a repository. A repository's popularity is measured with the logarithmic value of its number of stars, and repositories with no stars are excluded for this analysis as they may be created simply for code storage or testing. This data subset includes the metadata and embeddings of 56k Python repositories and 36k Java repositories. The statistics in the metadata, except the number of stars, were used as baseline features. We trained a random forest estimator of 500 trees with different feature sets, including baseline features, the different embeddings, and their concatenations, and then compared their prediction performance on the test set comprising 20% of the data. For each feature set, the task was replicated ten times with shuffled train-test splits, and the final performance was determined by the mean values of explained variance scores (EV) and mean squared errors (MSE).

The results in Table 2 show that the best feature set for the Python repositories is the combination of import-based embedding, readme-based embedding, and baseline features, while the combination of contributor-network-based embedding, readme-based embedding, and baseline features outperforms others for Java. In general, we found that functionality information is more relevant to the popularity of repositories in Python than social information, regardless of whether it is combined with other features. On the contrary, social information from the co-contributor network is more valuable than functionality information for Java repositories' popularity and is even more informative than some key baseline features. Figure 3 contains a visualization of the ten most important features for the best models, for both Python and Java. These features reveal a clear sign of the importance of functionality and social context across languages. For instance, repositories with the highest values in Dimension 42 of the Python readme-based embedding space are related to convolutional neural networks (CNN), and repositories with the lowest values in the fourth dimension of the Java contributor-based space are affiliated with Netflix.

Features	Python		Java	
	EV	MSE	EV	MSE
baseline	0.2842	1.8001	0.2677	2.0408
import	0.1461	2.1495	0.1045	2.4996
readme	0.1798	2.0628	0.1989	2.2321
network	-0.0357	2.6093	0.3295	1.8685
import+readme	0.2274	1.9423	0.2310	2.1419
import+network	0.1274	2.1960	0.3280	1.8719
import+baseline	0.3508	1.6334	0.3252	1.8815
readme+network	0.1590	2.1149	0.3679	1.7589
readme+baseline	0.3780	1.5635	0.3833	1.7174
network+baseline	0.2795	1.8128	0.3755	1.7396
import+readme+network	0.2145	1.9745	0.3719	1.7478
import+readme+baseline	**0.3822**	**1.5530**	0.3795	1.7279
import+network+baseline	0.3331	1.6775	0.3723	1.7481
readme+network+baseline	0.3592	1.6107	**0.4072**	**1.6495**
import+readme+network+baseline	0.3692	1.5854	0.4037	1.6592

Table 2. Popularity prediction results

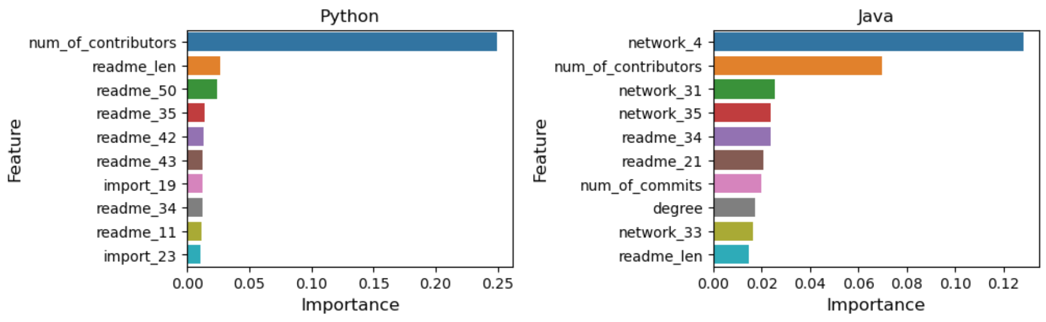

Fig. 3. Feature importance for the most important ten features of the best models (network means the contributor-based embeddings, and the numbers are the indices of dimensions)

4 DISCUSSION OF RESULTS

Python was released with an emphasis on code readability and conciseness. Tim Peters' "Zen of Python" [21] illustrates the essence of Python's philosophy: "Beautiful is better than ugly. Explicit is better than implicit. Simple is better than complex. Complex is better than complicated. Flat is better than nested. Sparse is better than dense. Readability counts." Meanwhile, the philosophy of Java highlights the independence and completeness of code, stating "Write Once, Run Anywhere" [2]. This discrepancy in philosophy is reflected as substantial differences in coding practices. For instance, Python allows coders to declare variables without specifying their data types, but Java requires it explicitly. Besides, including external libraries in Java requires coders to manually download source files and add their paths to the IDE. While in Python, this can be achieved with much less effort by using widespread package management commands, such as the "pip install" command. These differences contribute to their divergence in application scenarios. Python's flexible and succinct style meets the demand for areas like data science, where researchers highlight the reusability of code, and the invention of IPython and Jupyter notebook further empowered Python for data analysis by enabling users to arrange code, visualization, and annotations in the same place. In comparison, Java is more suitable for programming at an engineering level, like those for mobile applications and software products. From an economic perspective, the "consumers" in the Python market are mainly composed of peers who also write Python code and wish to import other Python libraries which implement useful features, whereas Java developers generally face more passive end-users who ask for handy software without the need to do any coding.

Their differences help contextualize the findings from our study. The flexibility and convenience of using resources from external libraries prompt Python programmers to import more in their code and make users more adventurous when exploring different functional areas in development. Moreover, the familiarity with different functional areas of code gives a programmer a greater advantage in the Python "market," enabling them to write code that will be popular with more "consumers": other programmers. Thus, we see that Python developers on GitHub value the diversity in contribution history and are less likely to just stay in one area (organized by functionality). Also, Python repositories gain popularity more because they realized functions favored by other Python users. Conversely, competitive Java repositories are probably monopolized by tech-companies or platforms, close-knit hubs of proficient Java developers, and the implementation of a certain function is often bound to the sources of these platforms. Therefore, we see a high consistency between the functionality space and the social space in the Java community, and social affiliation is more important for the popularity of Java repositories. Switching between different functionality is usually costly for Java developers as they need to learn paradigms for new frameworks, so most of them focus on one or few related areas, which can bring them the merit to serve the end-users with high-quality products.

5 CONCLUSION

Inspired by the Sapir-Whorf hypothesis, this research is the first to empirically demonstrate the heterogeneity in social coding behaviors across programming languages. Specifically, we examined the differences between the Python and the Java community on GitHub. Through sub-studies of socio-functional mapping and programmers' contribution diversity, we identified that the Python programmers are less socially organized by repositories' functionality than the Java coders, and they attach more values to the diversity in contribution experience. Besides, in the sub-study of repository popularity, we found that the popularity of Python repositories is more revealed by information from the function-based embeddings, while social context is more important for that of Java repositories. Finally, we discussed how these differences relate to the philosophies of Python and Java, which provides the underlying ideological motivation for our findings.

While our research provides evidence for the existence of linguistic relativity in programming, it does not directly illustrate the difference in how programmers use these programming languages, i.e., how they write code. We believe that future studies to analyze how programmers frame their code for different purposes, choose different programming languages for different tasks, or adopt the idiomatic structures of one programming language while coding in another would help to further investigate the hypothesis. Additionally, extending our methods and research to the relationships between programming languages beyond Java and Python might enable the development of broadly applicable tools to help programmers identify the best language & community for their particular long-term goals.

REFERENCES

[1] Laura M Ahearn. 2021. *Living language: An introduction to linguistic anthropology.* John Wiley & Sons.

[2] Ken Arnold, James Gosling, and David Holmes. 2000. The Java Programming Language Addison.

[3] Natércia A Batista, Michele A Brandão, Gabriela B Alves, Ana Paula Couto da Silva, and Mirella M Moro. 2017. Collaboration strength metrics and analyses on GitHub. In *Proceedings of the International Conference on Web Intelligence*. 170–178.

[4] Angie Boggust, Brandon Carter, and Arvind Satyanarayan. 2019. Embedding comparator: Visualizing differences in global structure and local neighborhoods via small multiples. *arXiv preprint arXiv:1912.04853* (2019).

[5] Hudson Borges and Marco Tulio Valente. 2018. What's in a github star? understanding repository starring practices in a social coding platform. *Journal of Systems and Software* 146 (2018), 112–129.

[6] Eleni Constantinou and Georgia M Kapitsaki. 2016. Identifying developers' expertise in social coding platforms. In *2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 63–67.

[7] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. 2012. Social coding in GitHub: transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 conference on computer supported cooperative work*. 1277–1286.

[8] Tapajit Dey, Andrey Karnauch, and Audris Mockus. 2021. Representation of developer expertise in open source software. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 995–1007.

[9] David Flanagan and Yukihiro Matsumoto. 2008. *The Ruby Programming Language: Everything You Need to Know*. " O'Reilly Media, Inc.".

[10] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 855–864.

[11] Junxiao Han, Shuiguang Deng, Xin Xia, Dongjing Wang, and Jianwei Yin. 2019. Characterization and prediction of popular projects on github. In *2019 IEEE 43rd annual computer software and applications conference (COMPSAC)*, Vol. 1. IEEE, 21–26.

[12] Eric von Hippel and Georg von Krogh. 2003. Open source software and the "private-collective" innovation model: Issues for organization science. *Organization science* 14, 2 (2003), 209–223.

[13] Kenneth E Iverson. 2007. Notation as a tool of thought. In *ACM Turing award lectures*. 1979.

[14] Paul Kay and Willett Kempton. 1984. What is the Sapir-Whorf hypothesis? *American anthropologist* 86, 1 (1984), 65–79.

[15] EF Konrad Koerner. 1992. The Sapir-Whorf hypothesis: A preliminary history and a bibliographical essay. *Journal of Linguistic Anthropology* 2, 2 (1992), 173–198.

[16] Quoc Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *International conference on machine learning*. PMLR, 1188–1196.

[17] Antonio Lima, Luca Rossi, and Mirco Musolesi. 2014. Coding together at scale: GitHub as a collaborative social network. In *Eighth international AAAI conference on weblogs and social media*.

[18] Collin McMillan, Mark Grechanik, and Denys Poshyvanyk. 2012. Detecting similar software applications. In *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, 364–374.

[19] Jeremiah Milbauer, Yutao Chen, Deblina Mukherjee, and James Evans. 2021. Representing Repositories in the Open Source Ecosystem. (2021). https://github.com/jmilbauer/RepoReps/ 7th International Conference on Computational Social Science.

[20] Leonid Perlovsky. 2009. Language and emotions: emotional Sapir–Whorf hypothesis. *Neural Networks* 22, 5-6 (2009), 518–526.

[21] Tim Peters. 2010. The zen of python. In *Pro Python*. Springer, 301–302.

[22] Radim Řehůřek and Petr Sojka. 2010. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. ELRA, Valletta, Malta, 45–50. http://is.muni.cz/publication/884893/en.

[23] Bart Theeten, Frederik Vandeputte, and Tom Van Cutsem. 2019. Import2vec: Learning embeddings for software libraries. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 18–28.

[24] Ferdian Thung, Tegawende F Bissyande, David Lo, and Lingxiao Jiang. 2013. Network structure of social coding in github. In *2013 17th European conference on software maintenance and reengineering*. IEEE, 323–326.

[25] Yun Zhang, David Lo, Pavneet Singh Kochhar, Xin Xia, Quanlai Li, and Jianling Sun. 2017. Detecting similar repositories on GitHub. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 13–23.

[26] Nikolas Zöller, Jonathan H Morgan, and Tobias Schröder. 2020. A topology of groups: What GitHub can tell us about online collaboration. *Technological Forecasting and Social Change* 161 (2020), 120291.