How Do Contributors Impact Code Naturalness? An Exploratory Study of 50 Python Projects

Thanadon Bunkerd*, Dong Wang†, Raula Gaikovina Kula†, Chaiyong Ragkhitwetsagul*,
Morakot Choetkiertikul*, Thanwadee Sunetnanta*,Takashi Ishio†, and Kenichi Matsumoto†
*Faculty of Information and Communication Technology (ICT), Mahidol University
†Nara Institute of Science and Technology (NAIST)
Email: {thanadon.bun, chaiyong.rag, morakot.cho, thanwadee.sun}@mahidol.ac.th
{raula-k, ishio, wang.dong.vt8, matumoto}@is.naist.jp

***Abstract*—Recent studies have shown how software is comparable to natural languages, meaning that source code is highly repetitive and predictable. Other studies have shown the naturalness as indicators for code quality (i.e., buggy code). With the rise of social coding and the popularity of open source projects, the software is now being built with contributions that come from contributors from diverse backgrounds. From this social contribution perspective, we explore how contributors impact code naturalness. In detail, our exploratory study investigators whether the developers' history of programming language experience affects the code naturalness. Calculating the code naturalness of 678 contributors from 50 open-source python projects, we analyze how two aspects of contributor activities impact the code naturalness: (a) the number of contributors in a software project, (b) diversity of programming language contributions. The results show that the code naturalness is affected by the diversity of contributors and that more collaborative software tends to be less predictable. This exploratory study serves as evidence into the relationship between code naturalness and the programming diversity of contributors.**

***Index Terms*—code naturalness, programming language diversity, developer experience.**

I. Introduction

The use of language models is now common in the Software Engineering domain, as a means to understand how the code is written. For example, the naturalness of code refers to the repetitive and predictable nature of the code in a project [18]. Several works have shown how naturalness is useful for refactoring, finding buggy code and so on. Language modeling has revealed power-law distributions and an apparent 'naturalness' of software source code [15].

With the rise of social coding and commonplace of collaborative platforms like GitHub[1], Gitlab[2] and BitBucket[3], contributions to software projects (especially open source projects) is more collaborative with contributions that are diverse. For example, contributions come from a diverse range of contributors, with different backgrounds and their experience in programming ability. A number of studies have shown that such open-source software (OSS) projects have significant advantages. As stated by Khanjani and Sulaiman [11], OSS is more flexible in term of productivity and increases motivation and performance, moreover, it allows faster bug detection and error resolution compared to closed source software (e.g. proprietary software projects). For example, Bavota and Russo [2] found that in the code review process of OSS projects, more eyes involved are more likely to reduce the chance of inducing bug fixes.

Software development involves a variety of human-intensive activities especially for collaborative platforms nowadays. Developers (i.e., humans) play an important role in the success of a software project [7]. There have been many prior social-technical studies performed on developer-related factors such as developer experience, the role and the social network in projects. Findings have shown that developers experience constitutes a key factor that needs to be carefully considered during maintenance tasks [4]. For instance, Casalnuovo et al. [6] found that past social connections combined with prior experience in languages dominant in the project lead to higher productivity both initially and cumulatively. For the role factor, Gilal et al. [8] investigated three independent role variables: team leader role, personality types, and gender. They claimed that the personality types of software development team roles fluctuate by gender type. For the social network factor, Posnett et al. [14] found that more focused developers introduce fewer defects than defocused developers. Yet, there has been no work that has looked at the diversity of programming language contributions as a quantitative metric. We propose the term *'diversity of programming language contributions'* (PLDiv) as a metric to understand how diverse developer contributions to projects with different programming languages.

In this paper, the research gap is that we would like to explore how contributors impact the code naturalness. Concretely, we would like to explore how the diversity of the programming experience of contributors affects the naturalness of software. In an exploratory study of 50 python projects, we study the relationship between the code naturalness and the diversity of their existing contribution activities. In detail, we propose a simple measure to calculate the diversity of programming language of contributors, using the history of their other contributions. We then form these two aspects as research questions to help guide our study:

[1]https://github.com/
[2]https://gitlab.com
[3]https://bitbucket.org

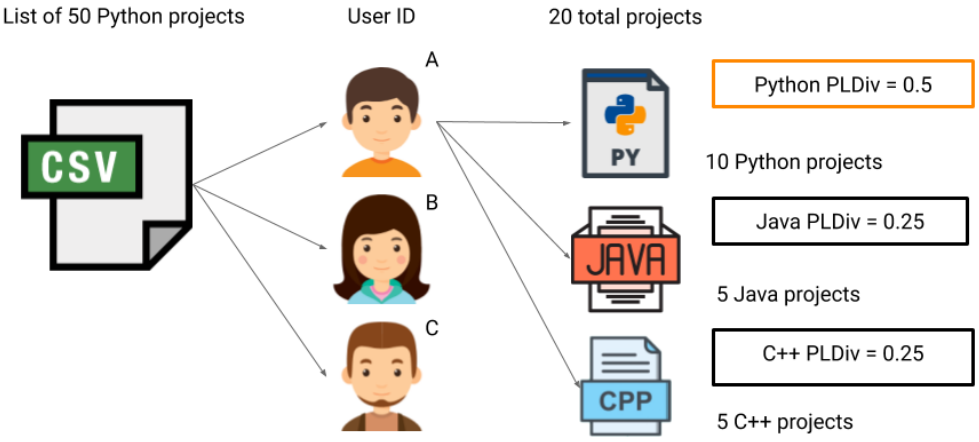

Fig. 1: Diversity of programming language contributions. This is the concept for the PLDiv metric that our paper proposes.

- **(RQ_1): *Does the number of contributors impact code naturalness?*** Our assumption is to test whether the degree of unique contributors has an impact on the code naturalness.
- **(RQ_2): *Does the diversity of programming language contributors impact code naturalness?*** Our assumption is to test whether programmers with a stronger contribution to a particular programming language (i.e., python) will tend to increase code naturalness of a project.

Results show that the code naturalness is affected by the diversity of contributors. We find that projects in which contributors have Python programming experience in the medium level (between two to seven contributors) tend to be less predictable code. Furthermore, projects with a large number of contributors tend to be less predictable code. The contribution of this paper is as follows:

- proposed metric to understand the diversity of contributions in contemporary software development projects.
- evidence that indicates the relationship between code naturalness and contributors.

II. Diversity of Programming Language Contributions (PLDiv)

Figure 1 depicts our metric used to calculate the diversity of programming language contributions. In detail, the contributor PL diversity (PLDiv) value of a project in a specific programming language (x) is computed as follows:

$$PLDiv(x) = \frac{1}{N}\sum_{i=1}^{N} l(x)_i / L_i \quad (1)$$

where $l(x)_i$ is a number of projects that a contributor i involves in a specific language (x), L_i is a total number of projects that a contributor i involves with (in any programming language), and N is a number of contributors in a project.

As shown in the figure, user ID *A* has a python experience of 0.5 (i.e., out of all their 20 projects, 10 are python projects, 5 are written in Java and 5 in C++). The normalized rate allows for a non-bias measure of developers regardless of their number of contributions and range of different programming language experiences.

III. Experiment Setup

To evaluate our proposed measure, we set up the following experiments to extract code naturalness.

A. Data Sources

We create our source code corpus from python open source projects collected from GitHub. As a starting point, we adopted the Python projects used in a prior study [10] and sampled 50 projects. To ensure that we processed tokens (i.e., code token) from each of the 50 projects, between 20M and 25M tokens. For all the projects, we examine only the master branch. A summary for each programming language project history is shown in Table I.

In the end, we were able to extract 678 contributors. Then using that as a starting point, we then track each contributor from each project, calculating the PLDiv for each. Detailed statistics of the contributors' information are shown in Table II.

B. Classification of Projects

As a baseline measure and in order to understand whether the number of contributors impacts the code naturalness (RQ_1), we classify the projects into three groups:

TABLE I: Descriptive statistics of the 50 projects in our dataset

Type	Min	Max	Mean	SD	Median
The Number of Contributors	1	566	22	81	3
The Number of Commits	22	22,473	1,862	3,402	907

Statistics for Contributor Commits	
Avg. # Projects Contributed	89.97 proj.
Avg. # Unique Languages	6.24 lang.
Avg. Commit Size	87.89 commits

TABLE II: Statistics of the PLDiv for a contributor

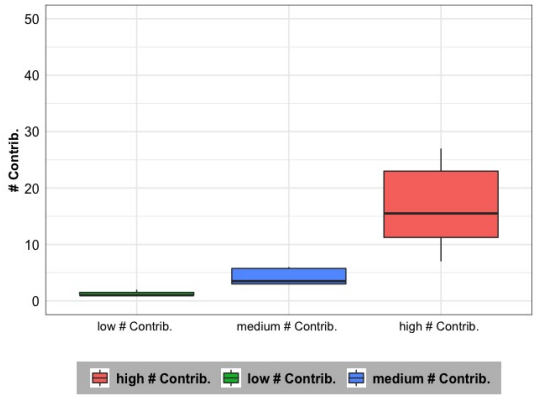

Fig. 2: Classifying projects based on number of contributors

- ***high # Contrib. projects*** - projects that have a high number of contributors.
- ***medium # Contrib. projects*** - projects that have a medium number of contributors.
- ***low # Contrib. projects*** - projects that have a few number of contributors.

In order to understand how the PLDiv impacts the code naturalness (RQ_2), we classify the same projects from RQ1 into three groups:

- ***high PLDiv projects*** - projects that have contributed to more python projects compared to projects written in other programming languages.
- ***medium PLDiv projects*** - projects that have contributed to other programming projects as much as python projects.
- ***low PLDiv projects*** - projects that have contributed the least to python projects compared to projects written in other programming languages.

Shown in Figure 2 and 3, we use Quantile-based discretization technique to make the three groupings. We use the python library `pandas.qcut`[4] to equally divide the data into three groups based on their distributions. *high PLDiv (0.6∼1.0)*, *medium PLDiv (0.4∼0.6)* and *low PLDiv (0∼0.4)*.

[4]https://pandas.pydata.org/pandas-docs/version/0.23.4/generated/pandas.qcut.html

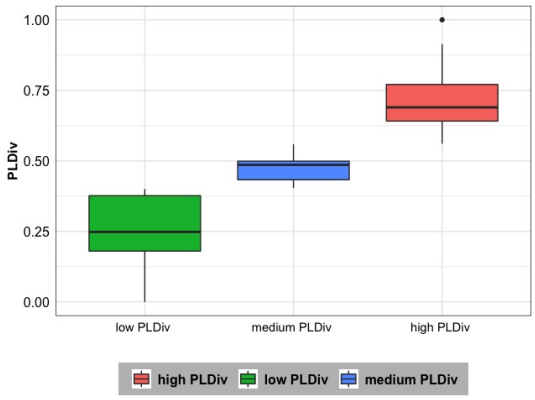

Fig. 3: Classifying projects based on PLDiv

In the case of the number of contributors per project, we use the same Quantile-based discretization technique: *high #Contrib* is a project with over 7 contributors, *medium #Contrib* is a project that has three to seven contributors, and *low #Contrib* is a project with less than or equal to 2 contributors.

C. Building the Language Model

Inspired by Rahman et al. [15], we use the same tools and methodology. For the source code tokenization, we use our python tokenizer[5], It uses a lexicalize based on ANother Tool for Language Recognition (ANTLR) and Anoyomouse/Antlr4-Grammar-JavaScript-Py. Then we merge all the lexicalized files to create a corpus for each file. Then we merge the processed files to create our final corpus for each of the 50 projects.

To calculate the entropy, a single corpus is split into 10 folds. Ten-fold cross-validation is used with the probability estimated from 90% of the data and validated on the remaining 10 %. The results are averaged over the 10 test folds.

We use MIT Language Model (MITLM) toolkit[6] to calculate the entropy for each data set. MITLM uses techniques for n-gram smoothing to deal with unseen n-grams in the test fold (see [15] for further discussion). We calculate the entropy for token sequences, i.e. n-grams, from 1-grams to 10-grams for each corpus.

D. Analysis

Similar to the work of Rahman et al. [15], we visualize the entropy against the n-grams within each analysis of the results. For RQ1, we employ the same analysis, however, we use the

[5]https://github.com/Ikuyadeu/CodeTokenizer
[6]https://github.com/caseycas/CacheModelPackage

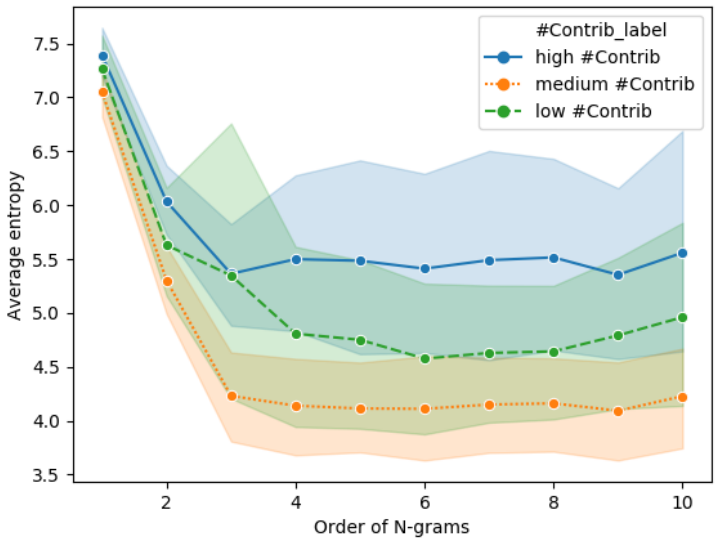

Fig. 4: Average entropy comparing with total number of contributors in Python. The result suggests that projects that are more collaborative tend to have less predictable code.

high # Contrib projects, medium # Contrib projects, and low # Contrib projects for our analysis. The lower the score of the entropy, means the higher likelihood for code naturalness. Then for RQ2, we use the same analysis of the contributors' classifications of High PLDiv, medium PLDiv, low PLDiv for RQ1.

IV. Results and Discussion

In this section, we present the results and then answer the two research questions.

a) ***(RQ_1): Does the number of contributors impact code naturalness?****:* Figure 4 shows that projects with a higher number of contributors tend to have less predictable code. In contrast, the results show that projects with a medium number of contributors tend to have a more predictable code.

The results show that contributors do have an impact on code naturalness. Intuitively, the more collaborative a project is, the more it tends to have less predictable code. At this stage, we did not measure how contributors profiles (i.e., how active and how long they have been contributing). This is seen as interesting future work.

> **Summary:** The projects within the group of the high number of contributors tend to have less predictable code than the projects having low and medium size of contributors.

b) ***(RQ_2): Does the diversity of programming language contributors impact code naturalness?****:* Figure 5 shows that contributors with low PLDiv tend to belong to projects that have more code naturalness. In contrast, this also means that contributors that have medium PLDiv tend to belong to projects that have the lowest code naturalness.

The results show that how contributions impact the code naturalness is not trivial. From these results, we can summarize that maybe that GitHub projects tend to have code that is not completely repetitive. The result is indicative that collaborative software may have a range of naturalness, and, in fact, having more PLDiv might not be an impacting factor for understanding naturalness.

Thus, for future work, there could be other metrics and measures that are related to the naturalness, including the naturalness of the other projects, that might influence the naturalness of an existing project. Other interesting work could at the ecosystem level, to see whether the communities share more natural code as oppose to projects that contain a mixture of programming language projects.

> **Summary:** Projects in which contributors have Python programming experience in the medium level tend to have less predictable code.

V. Threats to Validity

We discuss two key threats to the validity of the study. The first threat is in terms of the construction of the experiment and threats on the tools. In the first threat, we understand that our classification is based on a sample of 50 projects. This might change with other types of projects. To mitigate this issue, we have used the statistical method to analyze the differences. We expect to improve the method in a more mature version of this work. The second threat is the generalization of the results. In this study, we only look at a case study of python projects. However, we cannot at this stage generalize our findings to other programming languages. We are confident that the

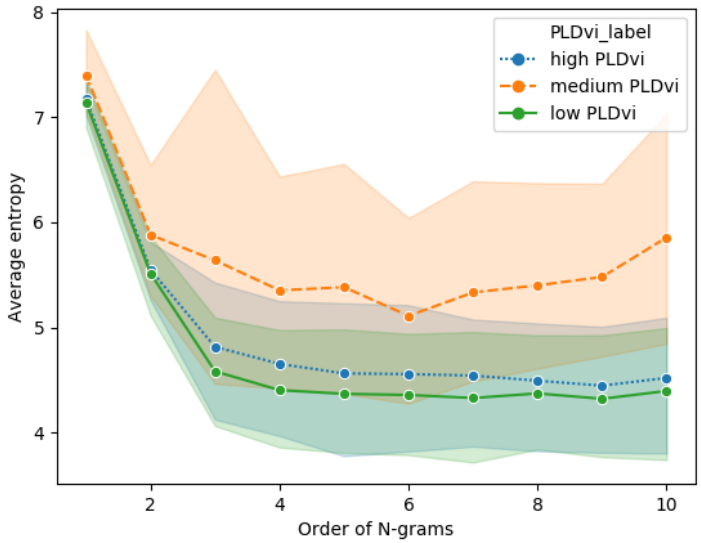

Fig. 5: Average entropy compared with contributors experience in Python. Projects with low python PLDiv have highly repetitive code.

collect 50 samples are enough in this exploratory study. The third threat is this study focuses on the diversity of programming language-specific among contributors in a project. The proposed definition of PLDiv thus mainly considers the number of contributors against the number of programming languages that they involved with. We, however, acknowledge that there are several factors (e.g. developer's expertise) that affect code naturalness which we aim to explore those factors in our future work.

VI. Related Work

In this section, we will present the related work focusing on software naturalness.

Research regarding naturalness has been widely studied in different aspects such as code completion and suggestion [9], the naturalness of names in code [5, 12] and summarization and concern location [16]. In detail, Ray et al. [17] analyzed the naturalness on buggy codes and found that code with bugs tends to be more entropic (i.e. unnatural), becoming less so as bugs are fixed.

There are a few studies regarding measuring developers' contributions and diversity in software projects. Ben et al. [3] use visual analysis to study contribution characteristics of programming in CraftBukkit project, an open-source software project. Gousios et al. [1] propose a hybrid model that combines traditional contribution metrics and the metrics mined from software repositories. The work that is closest to us is the study by Liang et al. [13]. The study investigates the effect of programmers' knowledge diversity (KD) and value diversity (VD) to the project performance using conflict theory. The authors found that KD is beneficial to the project outcome, while VD is harmful. Instead of studying the programmers' knowledge and value diversity, our study investigates the programmers' programming language contribution and its effect on the naturalness of the source code in software projects.

VII. Conclusion

Our exploratory study investigates whether the developers' history of programming language experience affects the code naturalness. Calculating the code naturalness of 678 contributors from 50 python projects, we analyze how two aspects of contributor activities impact the code naturalness. Results show that:

- Projects which are more collaborative with more contributors, tend to have less predictable code.
- Projects in which contributors have Python programming experience in the medium level tend to have less predictable code.

This exploratory study serves as evidence into the relationship between code naturalness and the programming diversity of contributors. For future work, it would be interesting to find out how naturalness evolves over time and how other metrics related to the programming language diversity could assist with understanding how code is written.

Acknowledgment

This research project was partially supported by the Faculty of Information and Communication Technology, Mahidol University and JSPS KAKENHI Grant Numbers 18H04094, JP18KT0013 and 17H00731.

References

[1] Measuring developer contribution from software repository data. In *Proceedings of the 2008 international*

workshop on Mining software repositories (MSR '08), page 129, 2008.

[2] G. Bavota and B. Russo. Four eyes are better than two: On the impact of code reviews on software quality. In *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 81–90, 2015.

[3] X. Ben, S. Beijun, and Y. Weicheng. Mining developer contribution in open source software using visualization techniques. In *Proceedings of the 2013 Third International Conference on Intelligent System Design and Engineering Applications*, pages 934–937, 2013.

[4] P. Bhattacharya, I. Neamtiu, and M. Faloutsos. Determining developers' expertise and role: A graph hierarchy-based approach. In *2014 IEEE International Conference on Software Maintenance and Evolution*, pages 11–20, 2014.

[5] D. Binkley, M. Hearn, and D. Lawrie. Improving identifier informativeness using part of speech information. In *Proceedings of the 8th Working Conference on Mining Software Repositories*, MSR '11, pages 203–206, 2011. ISBN 978-1-4503-0574-7.

[6] C. Casalnuovo, B. Vasilescu, P. Devanbu, and V. Filkov. Developer onboarding in github: The role of prior social links and language experience. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, pages 817–828, 2015. ISBN 978-1-4503-3675-8.

[7] T. DeMarco and T. Lister. *Peopleware: Productive Projects and Teams (3rd Edition)*. Addison-Wesley Professional, 3rd edition, 2013. ISBN 0321934113, 9780321934116.

[8] A. R. Gilal, J. Jaafar, M. Omar, S. Basri, and A. Waqas. A rule-based model for software development team composition: Team leader role with personality types and gender classification. *Information and Software Technology*, 74:105 – 113, 2016.

[9] S. Han, D. R. Wallace, and R. C. Miller. Code completion from abbreviated input. In *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering*, ASE '09, pages 332–343, 2009. ISBN 978-0-7695-3891-4.

[10] H. Hata, C. Treude, R. G. Kula, and T. Ishio. 9.6 million links in source code comments: Purpose, evolution, and decay. In *Proceedings of the 41st International Conference on Software Engineering*, ICSE '19, 2019.

[11] A. Khanjani and R. Sulaiman. The process of quality assurance under open source software development. In *2011 IEEE Symposium on Computers Informatics*, pages 548–552, 2011.

[12] D. Lawrie, C. Morrell, H. Feild, and D. Binkley. What's in a name? a study of identifiers. In *Proceedings of the 14th IEEE International Conference on Program Comprehension*, ICPC '06, pages 3–12, 2006. ISBN 0-7695-2601-2.

[13] T. Liang, C. Liu, T. Lin, and B. Lin. Effect of team diversity on software project performance. *Industrial Management & Data Systems*, 107(5):636–653, 2007.

[14] D. Posnett, R. D'Souza, P. Devanbu, and V. Filkov. Dual ecological measures of focus in software development. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 452–461, 2013.

[15] M. Rahman, D. Palani, and P. C. Rigby. Natural software revisited. In *Proceedings of the 41st International Conference on Software Engineering*, ICSE '19, pages 37–48, 2019.

[16] S. Rastkar, G. C. Murphy, and A. W. J. Bradley. Generating natural language summaries for crosscutting source code concerns. In *Proceedings of the 2011 27th IEEE International Conference on Software Maintenance*, ICSM '11, pages 103–112, 2011. ISBN 978-1-4577-0663-9.

[17] B. Ray, V. Hellendoorn, S. Godhane, Z. Tu, A. Bacchelli, and P. Devanbu. On the "naturalness" of buggy code. In *Proceedings of the 38th International Conference on Software Engineering*, ICSE '16, pages 428–439, 2016. ISBN 978-1-4503-3900-1.

[18] R. Robbes and M. Lanza. Improving code completion with program history. *Automated Software Engg.*, 17(2): 181–212, 2010. ISSN 0928-8910.