An Empirical Study to Investigate Collaboration Among Developers in Open Source Software (OSS)

Weijie Sun*, Samuel Iwuchukwu†, Abdul Ali Bangash‡ and Abram Hindle§
Department of Computing Science, University of Alberta
Edmonton, AB, Canada
Email: *weijie2@ualberta.ca, †siwuchuk@ualberta.ca, ‡bangash@ualberta.ca, §hindle1@ualberta.ca

***Abstract*—The value of teamwork is being recognized by project owners, resulting in an increased acknowledgement of collaboration among developers in software engineering. A good understanding of how developers work together could positively impact software development practices. In this paper, we investigate the collaboration habits of developers in project files by leveraging the World of Code (WoC) dataset and GitHub API. We first identify the collaboration level of developers within the project files, such as the source, test, documentation, and build files, using the Author Cross Entropy (ACE). From the results we find out that test files report the highest degree of collaboration among the developers, perhaps because collaboration is critical to ensure convergence of functionality tests. Furthermore, the source code files show the least degree of collaboration, perhaps because of code ownership and the complexity and difficulty in code modification. Secondly, given the widespread usage of the Python programming language, we investigate the Python code tokens that are more prone to change and collaboration. Our findings offer insights into the specific project files and Python code tokens that developers typically collaborate on in the open-source community. This information can be used by researchers and developers to enhance existing collaboration platforms and tools.**

***Index Terms*—OSS, WoC, development-practice**

I. Introduction

Open-source software (OSS) is a publicly accessible and freely distributed system, offering benefits such as improved security and freedom from vendor lock-in [1]. In order to be available for public use, OSS systems often require developers to collaborate on certain development activities, such as code refactoring, modification, build configuration, documentation, and writing of tests. Improved understanding of developer collaboration can result in the development of better collaborative platforms and tools. In this paper, we aim to understand how developers collaborate in the OSS environment on different project files and project code-tokens.

To investigate the trend of developer collaboration on different project files, we use Cross Entropy [2], a measure to realize the degree of developer involvement in a particular file. Furthermore, we use Author Cross Entropy (ACE) [3] to classify the degree of collaboration between the authors on a particular file. We investigate collaboration trends across 4 file types (source, test, build, and documentation files). After observing the developers' collaboration trends on project files, we investigate how developers collaborate over Python code tokens. We specifically focus on Python due to its widespread use in open-source software [4]. To obtain the project files, we randomly extracted $20,000$ collaborative GitHub projects from the World of Code (WoC) dataset [5].

To summarize, we investigate the following research questions:

RQ1: What is the difference in collaboration distributions (ACE) across 4 file types: Source Code, Build files, Tests, and Documentation?

RQ2: What tokens in Python are the most collaborative?

Our results report a high degree of developer collaboration in test files. This result indicates that developers usually require a high level of collaboration to write tests. On the contrary, the source code files show minimal collaboration, likely due to code ownership and how some authors contribute more than others. In addition to investigating the collaboration on files, we investigate collaboration on Python code tokens as well. We first extracted code tokens that developers had collaborated on, and then we used lexical analysis to compare the changed version of the tokens with their unchanged version. We make this comparison to determine the degree of collaboration between the developers. From the comparison, we find out that certain tokens report high collaboration, such as `identifiers`, `strings`, `imports`, and `comments`. In contrast, certain tokens report low collaboration, such as `operators`, `asserts`, and `whitespaces`.

II. Methodology

The methodology of this study is depicted in Figure 1. First, we find all projects that have more than a single author and are available on Github leveraging the *World of Code* (WoC) dataset [5]. Then we randomly sample $20,000$ projects from that subset and use GitHub API to retrieve all lines of changes in all commits, as we face computational limits on using the GitHub API. Then for RQ1, we calculate author cross entropy distributions of files based on commits, split the analysis by file type, and then compare author cross entropy of file types with the Mann-Whitney U rank test [6]. For RQ2 we filter the $20,000$ projects further, selecting those that have Python files with multiple contributors. We then count and calculate the percentage of tokens added in commits and compare it against the frequency of tokens overall.

a) Author Identity: In WoC [5], we found $125,154$ unique tuples of authors and projects. To resolve authors who have multiple email addresses (aliased authors) we adopt the

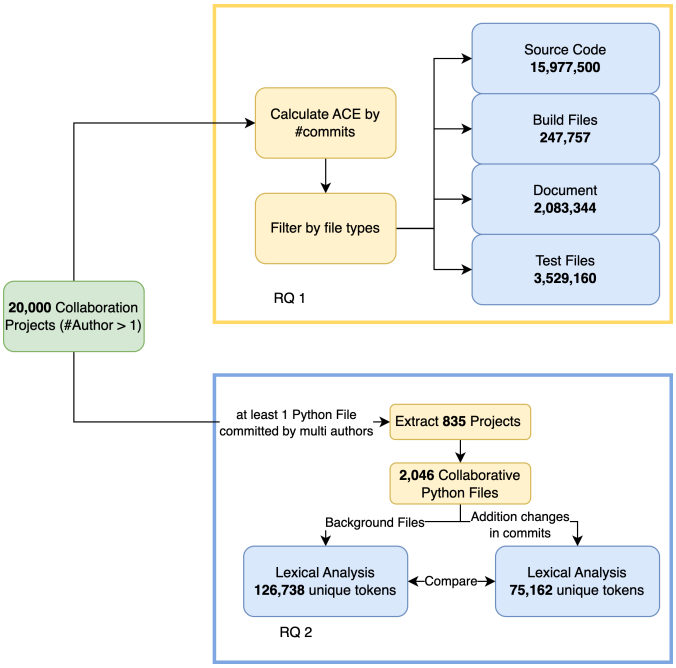


Fig. 1. The methodology to extract projects and analyze their developers' collaboration.

methodology of Tanner *et al.* [7], which maps author identity from similar author emails to author names. However, without first and last names we merge aliased authors, within the same project, according the following three scenarios:

- Author A's author name and Author B's author name are the same. e.g. *<Alan Turing alan@example.com> and <alan turing AlanT@example.com>*
- Author A's author name and Author B's email prefix[1] are same. — e.g. *<Alan Turing alan@example.com> and <A Turing AlanTuring@example.com>*
- Author A's email prefix and Author B's email prefix are the same. — e.g. *<Alan Turing alan@example.com> and <A Turing alan@email.com>*

As a result, we found $15,080(12\%)$ duplicated author-project tuples. This reduced the number of author-project tuples from $125,154$ to $110,074$.

b) File Classification: To identify the degree of collaboration in different file types, we classify the files into four majority file types. We considered the use of file types since it provides better details on the task type involved in the collaboration. Then we used Hindle *et al.*'s [8] method, which classifies file types by matching the file extensions and filenames. Then, we classified the files into four major file types: Source Code (S), Build files (B), Tests (T), and Documentation (D).

c) Author Cross Entropy: To estimate the collaboration level of files, we used the *Author Cross Entropy (ACE)* method from Taylor *et al.* [3] to characterize the distribution of the authors' contributions within each file type. We estimate the ACE at the granularity of a number of commits. For example, if majority of file-commits are from a single author, then the file has a lower ACE. Higher ACE also indicates the files were created collaboratively in a fair or equal manner (a flat distribution). Where n is number of authors and $P(i)$ is the frequency (between 0 and 1) of author i's commits on a file, ACE is defined by the following equation:

$$\text{Author Cross Entropy} = -\sum_{i}^{n} P(i) \cdot \log P(i) \qquad (1)$$

d) Comparison: We want to compare if the collaboration level is different between file types. Since file types have different sizes, and there are no assumptions for data distributions, we used the Mann-Whitney U rank test [6] to compare the pair ACE distribution for each file type. We have applied two different hypothesis: (1) H0: the two distributions are equal; H1: the two distributions are not equal (2) H0: distribution 1 is less than distribution; H1: distribution 1 is greater and equal than distribution 2. Then, in each hypothesis we consider if p-value < 0.05 means we will reject null hypothesis and choose alternative hypothesis.

e) Lexical Analysis: For RQ2, we want to discover the collaboration level of per token in Python code. We use lexical analysis libraries [9] to extract 9 different types of Python tokens:

- **Identifier**: Identifiers are the name given to variables, classes, methods, e.g., `add_argument`, `os`, `join`, etc.
- **String**: The words are surrounded by either single quotation marks, or double quotation marks, e.g., *Menu.html*, *hello*, etc.
- **Comment**: Comments are started with a # or a comment string of 3 quotes (`'''`). Python will ignore them.
- **Operator**: Operators are used to perform operations on variables and values, e.g., +, -, =
- **Keyword**: Keywords are reserved words in Python in Python, e.g., `for`, `except`, `if`.
- **Number**: Number is a numeric literal, e.g., `1.1`, `10`, `-20.5`.
- **Decorator**: Decorator provides the functionality to wrap another function to extend the behaviour of the wrapped function and start with `@`, e.g., `@my_decorator`.
- **White Space**: Characters such as spaces, tabs, and newlines used in indentation, line-breaks, or spacing between operators and identifiers.
- **Other**: Other operators, e.g., `?, ~ .`

III. Results

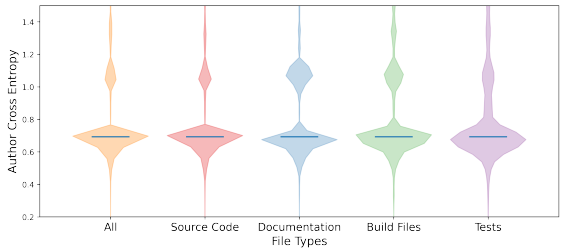
A. Collaboration Across File Types (RQ1)

Before calculating the collaboration distributions, we classify $21,837,761$ files from $20,000$ projects into 4 file types: Source Code, Build files, Tests, and Documentation. First we measure the distribution of collaborators per file type. Table I summarizes the count of files that are associated with the number of authors who have contributed to the file's type.

[1]Email prefix is the email content before @. e.g. alan is the email prefix of alan@example.com

TABLE I
ACE DISTRIBUTIONS OF ALL FILES, SOURCE CODES, DOCUMENTATIONS, BUILD FILES AND TESTS.

Authors	Source Code	Tests	Document	Build Files
1	12228517	2618221	1525821	195454
2	2743892	596494	348110	34587
3	600149	163073	173321	8493
4	140546	46019	16188	2606
5	81748	33161	7823	1862
6	31979	16785	3173	929
>7	150669	55407	8908	3826
SUM	15977500	3529160	2083344	247757



Then, we find that all four file types have a left-skewed distribution of the number of collaborators.

Secondly, we investigate the distribution of ACE scores per file type, shown in Figure III-A. Then, we used Mann-Whitney U tests [6] to compare the ACE distributions between each pair of file types. In each pair of Mann-Whitney U test results, if file type A shows significant greater than file type B (p-value ≤ 0.05), then we consider authors collaborate more in file type A than file type B. Our results show four file types from the most collaborative to least collaborative: Tests $>$ Documentation $>$ Build Files $>$ Source Code.

Code ownership of source code, even in OSS, is well documented [10], [11], thus it is unsurprising that source code is less collaborative than file types such as tests. Perhaps tests are more collaborative due to the involvement of multiple stakeholders, as tests give immediate feedback if dependencies and expectations have been changed within the source code.

B. Collaboration Across Python Code Tokens (RQ2)

We want to explore the token distribution in *collaborative content* (content added or modified by multiple authors), with a specific focus on the added or modified tokens in each commit changes. To discover the collaboration features in Python files, we filtered 835 projects with at least one Python file committed by multiple authors of $20,000$ projects. In 835 projects, there are $2,046$ Python files committed by multiple authors. Moreover, we downloaded the latest version of these same $2,046$ Python files and did the lexical analysis for the entire content. Finally, we named these as *python background files* and compared them with *collaborative content*. Next, we used the WoC database [5] to find the project's url, commits, commit authors and changed files in each project. To find the collaborative line of code, we used GitHub API to retrieve the commit diff files.

We applied the lexical analyzer on added lines from commit changes, which consists of newly added lines and modified lines. We extracted a total $75,162$ unique Python tokens from collaborative content in $2,046$ Python files. To compare the collaborative content with Python background files, we extracted total $126,738$ unique Python tokens from the latest version of the same $2,046$ Python files.

Table II shows the Python token type distribution in both Python background files and collaborative content. Accordingly, authors collaborate the most in identifier of all token types, which is the 3rd rank in Python background files. Authors might frequently update the variable name, and function names while fixing bugs or refactoring. Moreover, we discovered that collaborators frequently update or add the `string` and `comment` tokens. In `string` token type, they make up 20.9% updated tokens in commit changes. Authors might update the string content in the source code for more comprehensive descriptions or proper words iteratively. `Comment` token type makes up 10.7% of collaborative content. To collaborate well with team mates, once authors make code modifications, they often add the comments to explain their changes. Finally, we realized the collaborative content ranks of `operator` and `whitespace` decreased compared with ranks in background files. Authors barely update or add `operator` tokens in collaborative content with only 8.95% compared with 35.98% in background files perhaps because operators are fundamental logic that might cause unexpected issues. Moreover, the percentage of `whitespace` tokens in collaborative content is 1.68%, while in background files it is 27.63%. In collaborative content, the majority of added or removed `whitespace` tokens are space characters between operators and identifiers. However, in background files, the types of `whitespace` include line breaks and indents. Collaborators may less frequently add or update line breaks and indents in their commits compared to the background files.

Python keyword token Table III shows that authors update the `import` keyword token more often in collaborative content. The rank of `import` increases from fifth in the Python background files to the first in the collaborative content. We believe there are two possible scenarios: (1) when developers are adding new features, fixing bug or optimizing code performance, they might import new modules or libraries; (2) when developers want to refactor code, they simplify the code by creating reusable functions. Therefore, moving functions to modules causes the need for the `import` statement in collaborative content. In both of the above scenarios, developers might add more lines including the `import` keyword. Furthermore, we find the `assert` has 0.01% in collaborative content but 1.09% in background files. According to previous literature [12], [13], using a high density of `asserts` improves the effectiveness of testing and improves code quality. In our study, the collaborative content has unexpected lower percentage of `assert` tokens compared with background files.

TABLE II
PYTHON TOKEN TYPES' COUNT, PERCENTAGE, AND RANK IN BOTH COLLABORATIVE CONTENT AND PYTHON BACKGROUND FILES.

Token Type	Count Collaborative Content	Percentage Collaborative Content	Collaborative Content Rank	Background Files Rank Background Files	Count	Percentage
IDENTIFIER	94657	48.63%	1	3	915385	24.21%
STRING	40688	20.90%	2	4	190866	5.05%
COMMENT	20836	10.70%	3	7	52482	1.39%
OPERATOR	17417	8.95%	4	1	1360274	35.98%
KEYWORD	10430	5.36%	5	5	145588	3.85%
NUMBER	6938	3.56%	6	6	64987	1.72%
WHITESPACE	3273	1.68%	7	2	1044772	27.63%
DECORATOR	236	0.12%	8	8	5325	0.14%
OTHER	182	0.09%	9	9	1480	0.04%

TABLE III
PYTHON KEYWORD TYPES' COUNT, PERCENTAGE, AND RANK IN BOTH COLLABORATIVE CONTENT AND PYTHON BACKGROUND FILES.

	Collaborative Content			Background Files		
Type of Keyword	Count	Percentage	Collaborative Content Rank	Background Files Rank	Count	Percentage
import	1196	11.47%	1	5	11333	7.78%
if	909	8.72%	2	1	25620	17.60%
def	832	7.98%	3	2	19541	13.42%
in	710	6.81%	4	4	13318	9.15%
from	698	6.69%	5	9	5665	3.89%
return	693	6.64%	6	3	14883	10.22%
for	613	5.88%	7	6	9020	6.20%
else	493	4.73%	8	7	6622	4.55%
not	436	4.18%	9	8	6334	4.35%
class	394	3.78%	10	14	2777	1.91%
and	369	3.54%	11	10	3577	2.46%
as	361	3.46%	12	16	2172	1.49%
try	290	2.78%	13	13	2812	1.93%
with	277	2.66%	14	20	1394	0.96%
is	272	2.61%	15	11	3214	2.21%

IV. RELATED WORK

The studies in collaborative software engineering have gained reasonable attention with a plethora of challenges arising in this niche.

Scacchi *et al.* [14] studied the collaboration practices and socio-technical relationships of Open Source Software. They observed that project leaders' instinct in managing software project issues might be to expand the team which doesn't always lead to a solution. Our study however focused on the granular aspect of collaboration by looking at software artifacts like files to gain accurate insights into the level of collaborative effort needed in creating and modifying projects.

Colazo *et al.* [15] leveraged data archives from OSS projects to identify the interrelations amongst collaboration patterns, quality, and efficiency of software products. Their results suggest that project managers should encourage internal collaboration. Our study however focused on the investigation of collaborations in software artifacts while helping project managers understand specific software artifacts that could require more collaborative efforts.

Hindle *et al.* [8] studied developer behavior, and the general nature of software processes by looking at activities in source, test, build, and documentation files that occurred when a software project is released. We used this classification of files in this study.

Tanner *et al.* [7] noted that authors could have multiple identities, and could triage bug reports to themselves. We saw duplicate author identities even in our datasets as some authors have multiple emails. We applied pre-processing techniques even in our samples to get single authors. The pre-processing method further highlights the value of relying on bug triage for identifying collaboration patterns in Open Source projects.

Avishkar *et al.* [16] leveraged Python tokens and LSTM models to learn the Python code suggestions. Their study leveraged findings from GitHub [17] to produce a code suggestion corpus for Python of about 41 million lines of Python code. We built on this intuition to identify tokens in Python code and identified the varying degrees of collaboration existing amongst them.

V. THREATS TO VALIDITY

We leveraged World of Code (WoC) [5] as our source of data and mined projects extracted from GitHub [17]. This means that replicating our methodologies outside the aforementioned platforms may not guarantee a similar outcome. Moreover, a high proportion of toy projects or student assignments causes sampling bias in the representation of all software development project collaborations. We only studied Python code thus we cannot guarantee similar results for other widely used programming languages.

Furthermore, we resolved author aliases, but might have missed some, or potentially combined aliases that were not aliases. Moreover, we adopted ACE as a means of measuring the degree of authors' involvement in the project which can be invalidated with a different mathematical function.

VI. Conclusion

We leveraged the World of Code (WoC) dataset [5] and GitHub API [17] in our study to extract and analyze $20,000$ random collaborated projects in a bid to identify collaboration patterns in Open Source Software tasks. We answered two research questions to guide our intuition in this study. Firstly, we started with identifying collaboration distribution patterns across different file types (*i.e.* source code, build files, test files, and documentation). This was done by leveraging the ACE of authors-changed files, with results showing a high level of collaboration in test files and source files being the least collaborated files. Secondly, we wanted to identify collaboration patterns across Python tokens by leveraging lexical analysis (RQ2). Our results show that tokens such as `identifiers`, `strings`, and `comments` are highly collaborative, while `operators`, and `whitespaces` are less collaborative. Python keywords like `import` are highly collaborative while keywords like `assert` are less collaborative. The replication kit of this work is made freely available [18].

Acknowledgement

This research was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) through their Discovery Grant program.

References

[1] Joseph Feller and Brian Fitzgerald, *Understanding open source software development*, Addison-Wesley Longman Publishing Co., Inc., 2002.

[2] Shie Mannor, Dori Peleg, and Reuven Rubinstein, "The cross entropy method for classification," in *Proceedings of the 22nd international conference on Machine learning*, 2005, pp. 561–568.

[3] Quinn C Taylor, James E Stevenson, Daniel P Delorey, and Charles D Knutson, "Author entropy: A metric for characterization of software authorship patterns," in *Third International Workshop on Public Data about Software Development (WoPDaSD08)*, 2008, p. 6.

[4] KR Srinath, "Python–the fastest growing programming language," *International Research Journal of Engineering and Technology*, vol. 4, no. 12, pp. 354–357, 2017.

[5] Yuxing Ma, Tapajit Dey, Chris Bogart, Sadika Amreen, Marat Valiev, Adam Tutko, David Kennard, Russell Zaretzki, and Audris Mockus, "World of code: Enabling a research workflow for mining and analyzing the universe of open source vcs data," *Empirical Software Engineering*, vol. 26, no. 2, pp. 1–42, 2021.

[6] Henry B Mann and Donald R Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *The annals of mathematical statistics*, pp. 50–60, 1947.

[7] Tanner Fry, Tapajit Dey, Andrey Karnauch, and Audris Mockus, "A dataset and an approach for identity resolution of 38 million author ids extracted from 2b git commits," in *Proceedings of the 17th international conference on mining software repositories*, 2020, pp. 518–522.

[8] Abram Hindle, Michael W Godfrey, and Richard C Holt, "Release pattern discovery: A case study of database systems," in *2007 IEEE International Conference on Software Maintenance*. IEEE, 2007, pp. 285–294.

[9] Github Account:nutbread, "lexical analysis libraries simple lexical analysis libraries for javascript and python," https://nutbread.github.io/lex/, March 2015.

[10] Michaela Greiler, Kim Herzig, and Jacek Czerwonka, "Code ownership and software quality: A replication study," in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, 2015, pp. 2–12.

[11] Matthieu Foucault, Jean-Rémy Falleri, and Xavier Blanc, "Code ownership in open-source software," in *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, 2014, pp. 1–9.

[12] Gemma Catolino, Fabio Palomba, Andy Zaidman, and Filomena Ferrucci, "How the experience of development teams relates to assertion density of test classes," in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2019, pp. 223–234.

[13] Gunnar Kudrjavets, Nachiappan Nagappan, and Thomas Ball, "Assessing the relationship between software assertions and faults: An empirical investigation," in *2006 17th International Symposium on Software Reliability Engineering*. IEEE, 2006, pp. 204–212.

[14] Walt Scacchi, "Collaboration practices and affordances in free/open source software development," *Collaborative software engineering*, pp. 307–327, 2010.

[15] Jorge A Colazo, "Collaboration structure and performance in new software development: findings from the study of open source projects," *International Journal of Innovation Management*, vol. 14, no. 05, pp. 735–758, 2010.

[16] Avishkar Bhoopchand, Tim Rocktäschel, Earl Barr, and Sebastian Riedel, "Learning python code suggestion with a sparse pointer network," *arXiv preprint arXiv:1611.08307*, 2016.

[17] GitHub, "Github rest api," https://docs.github.com/en/rest?apiVersion=2022-11-28, November 2022.

[18] Weijie Sun and Samuel Iwuchukwu, "Msr challenge oss collaboration replication kit," https://figshare.com/articles/software/MSR_challenge_Collaboration_OSS/22249156, March 2023.