An Empirical Analysis of Code Clone Authorship in Apache Projects

Reishi Yokomori
Dept. of Software Engineering
Nanzan University
Nagoya, Japan
yokomori@nanzan-u.ac.jp

Katsuro Inoue
Dept. of Software Engineering
Nanzan University
Nagoya, Japan
inoue599@nanzan-u.ac.jp

2023 IEEE 17th International Workshop on Software Clones (IWSC) | 979-8-3503-4442-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/IWSC60764.2023.00008

***Abstract*—Many studies have been conducted to identify various types of code clones with a focus on accuracy, scalability, and performance. However, there has been limited exploration into the nature of code clones. Even fundamental questions, such as whether authors who write many non-clone lines also tend to write many clone lines, or whether code snippets in the same clone set were written by the same author or different authors, have not been thoroughly investigated.**

In this paper, we explore such fundamental questions regarding code clone authorship. We analyzed Java files from 153 Apache projects on GitHub, with a focus on line-level granularity.

The analysis results showed that for 150 out of the 153 projects, the numbers of non-clone lines and clone lines contributed by each author are linearly correlated. We also found that two-thirds of the clone sets in all projects are primarily contributed to by single leading authors.

These results confirm our intuitive understanding of clone characteristics, even though no previous publications have provided empirical validation data from multiple projects. Since these results could assist in designing better clone management methods, we will explore the implications of developing an effective clone management tool.

***Index Terms*—authorship, git blame, single-leader clone set, multi-leader clone set**

I. Introduction

Research on code clones has been conducted since the 1990s, and since then, a large number of studies have been carried out and published [3], [11], [19]. Many of these studies have focused on algorithms, performance (such as recall and precision), and scalability of code clone detection tools. There have been also empirical studies on code clones, which mainly focus on the evolution of clones and the relation to fault proneness [13], [22]. These studies have only revealed a small portion of the characteristics of code clones. Knowing when, by whom, in what context, and how code clones were created is important information for managing code clones and supporting developers. As a part of understanding the characteristics of code clones, we became interested in investigating the authors of code clones.

Author information of code is a valuable resource for effective software maintenance. For example, Linares-Vásquez et al. proposed an approach to identifying expert developers with the author's information for software change requests [16]. Author information is also used for plagiarism detection and copyright infringement [23]. However, there is limited knowledge about the authorship of clones. Even the fundamental questions, such as whether authors who write many non-clone lines also tend to write many clone lines, or whether code snippets in the same clone set were written by the same author or different authors has not been thoroughly investigated. Previous research on clone authorship has primarily focused on a small set of projects, with an emphasis on change proneness and reusability [8], [18].

In this paper, we conduct an empirical study on the authorship of clones in the context of OSS project collections, specifically focusing on 153 Apache projects written in Java on GitHub. Our analysis for those target projects mainly focuses on two aspects: (1) correlation between the author's contributions to clone and non-clone lines, and (2) characteristics of single and multiple author clone sets.

Through the analysis, we found that the number of non-clone lines and clone lines contributed by each author have a linear correlation in most projects and one-third of clone sets are mainly contributed by multiple leading authors. These findings highlight the necessity of code clone management tools for the safe and consistent treatment of code clones.

Sec.II describes the terms and Sec.III discusses related work. Sec.IV presents two research questions and Sec.V shows the analysis result. Sec.VI discusses implications, and Sec.VII mentions the threats to the validity. Sec.VIII concludes the discussion with future work.

II. Description of Terms

A *code snippet*, or simply a *snippet*, is part of a source code file within a software system. Sometimes, we duplicate a code snippet by copying and pasting it, then modify the pasted part by changing the variable names or literals, within the same software system or across different ones.

A pair of two code snippets that are the same or similar is called a *clone pair*, and each snippet of the pair is referred to as a *code clone* or *clone* [11]. In this paper, we use the term 'clone' to mean a code snippet that has another code snippet of a clone pair in the same file or different files in the same project. We do not consider here inter-project clone pairs. The *length* of a clone (or *clone length*) is the lines of code (LOC) of the clone, which may include non-executable lines such as

DOI 10.1109/IWSC60764.2023.00008

comments or blank lines. A *clone set* (or clone class) is a set of code snippets in which any two elements form a clone pair. The *size* of a clone set is the number of elements (instances) of the clone set. The length of a clone set (or clone set length) is the average length of each clone in the clone set. A *clone line* is a line that is involved in a clone. Conversely, *non-clone line* is a line that is not involved in any clone.

A clone pair is generally categorized into type-1 to 4 by its similarity levels [11]. In this paper, type-1, type-2, and type-3 clones are our targets. Type-4 clones are interesting, but since their possibility of being created by the copy-and-paste actions is low, they were excluded from the scope of this study. The *author* of a line in a file is the author name given by the `git blame` command [6]. `git blame` is a command in Git that allows us to see who last modified each line. It shows each line of the file with a commit hash, the name of the author who last modified the line, and the date and time the modification was made. Only the name of the author is used here.

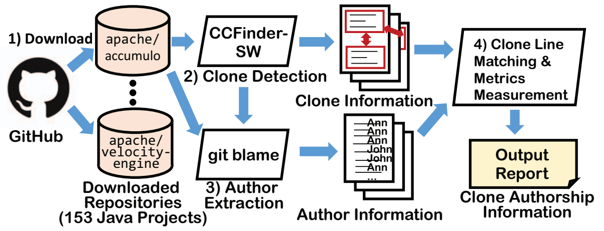


Fig. 1. Overview of Analysis Method

III. Related Work

A. Empirical Studies on Code Clones

Numerous studies on code clones have been performed and presented [3], [11], [19]. Among those, there were empirical studies on code clones. Goon et al. examined the ratio of code clones in C and C++ programs in OSS projects over time [7]. Barbour et al. analyzed faults in inconsistent clone changes [2]. Honda et al. investigated the change in code clone ratio from the initial development phase to maturity, finding that roughly 20% of the entire code consists of clones [9].

Harder investigated code clone authorship and changeability of clones for their clone detection tool as the analysis target [8]. His study found that the clone sets (classes) with a single author accounted for 66.3% of all clone sets, and that clone sets with multiple authors were twice as likely to be changed as those with a single author. While this study provides specific information on the authorship of clones for the targeted project, they do not give us a general idea of the relationship between clones and their authors for many other projects. Therefore, in this study, we conducted a detailed investigation on code clones and their relationship to the authors for many OSS projects.

Moriwaki et al. made an inter-project analysis of OSS projects and identified who reused whose source code across multiple repositories [18]. Although it provides an interesting perspective on the propagation of code snippets through projects and repositories, no detailed analysis of the authorship of clones inside a single project. In this paper, we are interested in the details of code clone authorship only within each project.

B. Code Authorship

Attributing code authorship is emerging research topics in, say, code security [12], plagiarism detection [15], and bug triaging [10]. These works mainly focus on identifying the authors of code snippets in source or binary forms by using comprehensive methods with the characteristics and statistics of codebases and their archives.

Using version management systems to identify the nature of clones can be seen in [14], in which they classify clones and identify copies from the originals. In our study, we rely on the feature of `git blame` [6] to attribute the authorship of code, although it might sometimes produce incorrect information [4]. We will discuss the threats of relying on the `git blame` command for code authorship attribution in Sec.VII.

IV. Research Approach

A. Research Questions

First, we examine the fundamental characteristics of authors who contributed to code clones.

RQ1: What is the correlation between the number of non-clone lines and clone lines contributed by each author?

This research question arises from an intuitive curiosity as to whether an author who writes many non-clone lines also writes clone lines. This might seem very obvious; however, there is little empirical data presented in previous literature to validate this assumption. We will analyze the contribution ratio of each author to clone lines compared to non-clone lines, and examine if there is a linear relationship between the clone and non-clone lines.

Next, we examine the authorship of clones within a single clone set.

RQ2: Are the clones in a clone set contributed by the same author, or are they contributed by different authors?

We will conduct an analysis of the authorship of clones within clone sets, by identifying clone sets contributed mainly by a single author. We will also examine the characteristic difference of clone sets contributed by single authors or multiple authors.

B. Method

Fig.1 shows an overview of our analysis method, which is described in the following steps.

1) First, we have downloaded repositories to analyze from GitHub. The target repositories are mentioned in the following Sec.IV-C. The following steps are executed for each repository.
2) Clone detection is performed by CCFinderSW [20], which is a token-based clone detector with a flexible tokenizer adaptable to many languages. It detects type-1 and 2 clones thoroughly and parts of type-3 clones (see the discussion in Sec.VII). CCFinderSW was chosen because it is written only

in Java and it works on various environments with high reliability and performance. We ran it with the default parameters, minToken=50 (the minimum number of tokens to detect) and tks=12 (the minimum number of different tokens in the clone fragments), and with a little higher rnr=0.5 (the minimum rate of non-repeating parts in the clone fragments, default=0.3) to reduce simple repeated statements. These parameters have been chosen to filter out smaller and accidental similar code snippets from the output. The output of CCFinderSW is composed of the target file information and the clone set information, both of which are used for the following steps.
3) `git blame` commands are executed for all source files to extract author information for each line from the repository.
4) The location of code clones obtained from CCFinderSW and the author information obtained from `git blame` are matched, and then the author information for the clone lines is obtained. Various statistics are measured and reported.

After completing step 1), the total execution time for steps 2)-4) to analyze the Apache Ant project was 226 seconds on a Ryzen 9-5900HX 32GB Windows 10 machine. Out of this total time, 28 seconds were spent on the clone detection step 2), 193 seconds were used for `git blame` in step 3), and 5 seconds were spent on clone line matching and metrics measurement in step 4), respectively.

C. Target Selection

We selected the Apache project's repositories on GitHub as the analysis target for the following reasons:

- Apache projects are open-source and typically supported by a community of contributors, making them suitable for our empirical investigation. We anticipated that the clones of these repositories would exhibit distinct characteristics compared to the previous research that was mostly developed by a single author.
- The Apache projects on GitHub encompass a diverse range of repositories, varying in size from small to large with a small to large number of contributors.
- Many Apache projects host their repositories on GitHub and are easily downloadable for coherent analysis.

We have identified and selected 166 Apache-owned projects on GitHub, meeting specific criteria including repository sizes between 100MB and 1GB, more than 100 followers, and written in the Java programming language. These constraints were used to match computing resource limitations and to exclude projects that are small or not popular. 13 projects were excluded because they had only a single author, leaving us with 153 projects for the subsequent analysis. To analyze each project, we chose the latest Java files in March 2023 from its project repository, excluding test files that contain the term ”test” in their file or path names.

Tab.I summarizes the characteristics of all 153 projects. For example, the size of the target Java files, measured in lines of code (LOC), ranges from 1.1K to 723.7K lines, with an average size of 125.9K lines. The ratio of clone lines to non-clone lines ranges from 4.2% to 178.0% with an average of

TABLE I
BASIC STATISTICS OF ALL 153 PROJECTS

Repositories (153 in total)	apache/accumulo ~ apache/velocity-engine
Repo's size	0.66- 3660 (74.8) MB
Number of target Java files (excl. test files)	7-3770 (791.5)
Total lines of target Java files	1.1-723.7 (125.9) K
Clone line ratio to total	4.1-64.1 (18.5)%
Non-clone line ratio to total	35.9-95.9 (81.5) %
Clone lines ratio to non-clone lines	4.2-178.0 (24.6) %
Number of clone set	4-11358 (1274)
Average clone length	9.1-45.7 (17.3)
Average clone set size	2.1-15.2 (3.3)
Num. of different authors in total lines	3-368 (56.0)
Num. of different authors in clone lines	2-225 (34.5)
Num. of different authors in non-clone lines	3-361 (55.0)

(The values inside parentheses are the average for all projects)

24.6%, showing the existence of projects with clone lines more than non-clone lines ($> 100\%$).

V. RESULT

As an example of a single project analysis, we will first present the results of the Apache Ant project, followed by the analysis of the set of all projects[1].

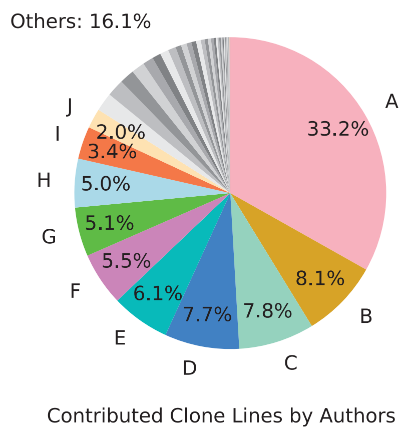
A. RQ1: What is the correlation between the number of non-clone lines and clone lines contributed by each author?

(Ant Project) The repository size of Ant is 97.2MB and the total LOC for all Java files is 235,939 with 33,064 clone lines (14.0%). There are 87 different authors in the total lines, with 86 in the non-clone lines and 59 in the clone lines in the target Java files. Fig.2 shows the ratio of the contributed clone lines (upper figure) and non-clone lines (lower figure) contributed by each author whose names are anonymized by symbols A, B, C, ...

As can be seen from these figures, although there are some differences in the order of contribution ratio, the distribution of the authors seems similar. This suggests that the contributions of the non-clone lines and clone lines by each author would be strongly correlated. We plotted the amount of author contribution to non-clone lines and clone lines as shown in Fig.3 where each dot represents one author with the number of contributed non-clone lines on the x-axis and that of contributed clone lines on the y-axis. Since this plot shows a strong linear correlation between non-clone and clone lines by authors, we fitted a linear regression as seen in Fig.3 where the coefficient is 0.164 and the coefficient of determination R^2 is 0.96. The regression coefficient of 0.164 indicates that authors are contributing to the creation of clone lines that account for 16.4% of non-clone lines.

To determine if two variables (the number of non-clone lines and the number of clone lines by each author) are linearly correlated, we calculated the Pearson correlation coefficient and obtained a p-value of 0.0. This means that we can reject

[1]All data can be seen from https://tinyurl.com/4d28dtev



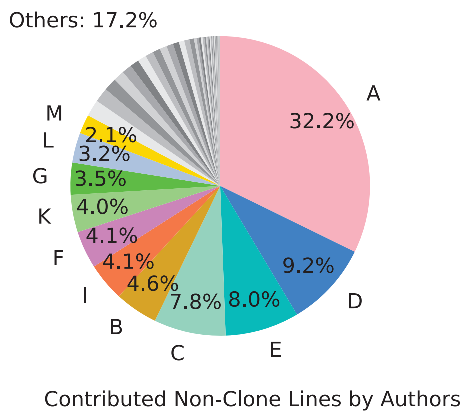


Fig. 2. Ratio of Contributed Clone and Non-Clone Lines by Authors

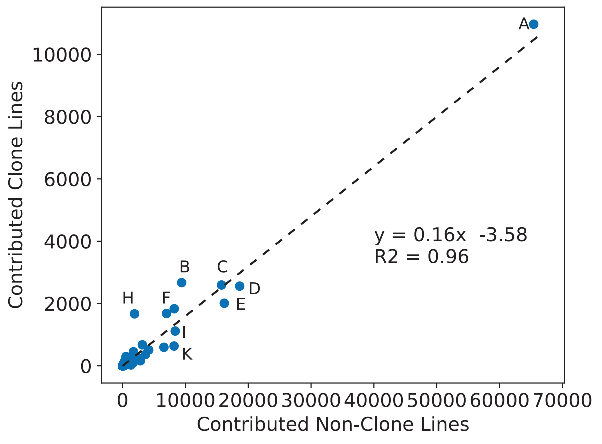


Fig. 3. Correlation of Author Contribution to Clone and Non-Clone Lines in Ant

the null hypothesis that the two variables are not linearly correlated, and therefore, the linear regression is validated.

(All Projects) We are interested in determining whether other projects exhibit a linear correlation between the number of non-clone lines and clone lines contributed by each author. As was done for the Ant project, we calculated the Pearson correlation coefficients to test the linear correlation between the two variables for all 153 projects.

The result shows that the null hypotheses were rejected for 150 projects with p-values less than 0.05. This means that, with three projects being the exception, we can say that authors who contribute to many non-clone lines also tend to contribute to many clone lines, and the number of clone lines contributed by an author can be simply predicted by the number of non-clone lines contributed by the same author.

Answer to RQ1: For 150 projects out of all 153 projects, the numbers of non-clone lines and clone lines contributed by each author have a linear correlation. In the Ant project, each author created clone lines at a rate of 16.4% of the non-clone lines.

B. *RQ2: Are the clones in a clone set contributed by the same author, or are they contributed by different authors?*

To perform this analysis effectively, we define two types of clone sets: *single-leader* clone set and *multi-leader* clone set. For a code snippet s, the *leading author* or simply *leader* l_s of s refers to the author who has committed the most lines in s, i.e., the top contributor in the snippet[2]. For a clone set C with clones $c_1, c_2, \ldots, c_n$ if all leaders of $c_1, c_2, \ldots, c_n$ are the same, then we say C is a single-leader clone set. Otherwise, it is called a multi-leader one.

The leader is the most contributing author. A single-leader clone set means that the leaders of each clone in the clone set are the same, even if there are different authors for a small portion of the clones. The reason for using leader to investigate the authorship of clones is that we want to know an overview of the dominant authors of clones, without being affected by minor contributions.

(Ant Project) Firstly, we analyze the proportion of single-leader and multi-leader clone sets in the Ant project. Out of exactly 1,000 clone sets found in the project, 527 (52.7%) are classified as single-leader, while 473 (47.3%) are multi-leader. Therefore, it can be concluded that approximately half of the clone sets in the project have multiple leaders. Fig.4 shows the breakdown of the number of leaders in each clone set, and it is clear that most multi-leader clone sets have two leaders.

Next, we conducted an investigation into the characteristics of single- and multi-leader clone sets in the Ant project. The lengths of the single- and multi-leader clone sets were measured and depicted as box plots in Fig.5. The average lengths of the single- and multi-leader clone sets were found to be 19.9 and 23.9 lines, respectively, with median values of 16 and 21 lines, respectively. The Mann-Whitney U-test revealed

[2]Note that if there are two or more such authors with the same number of committed lines, the leader is randomly selected from them.

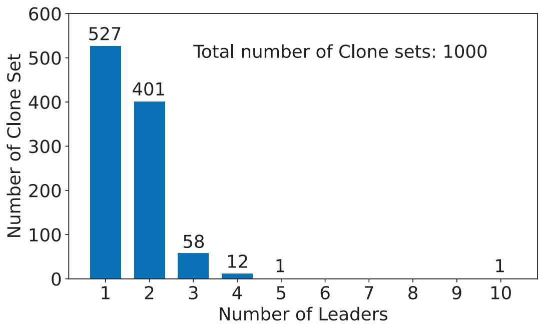


Fig. 4. Number of Leaders in Clone Set in Ant

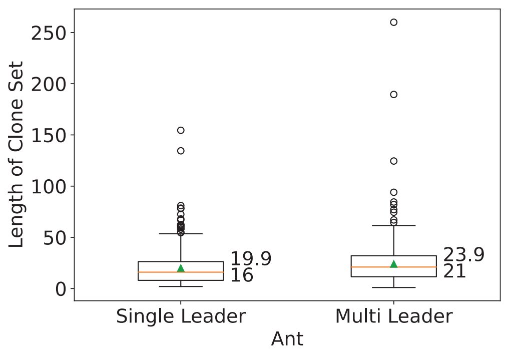


Fig. 5. Length of Single and Multi-Leader Clone Sets in Ant

a significant difference in clone length ($p = 2.14E - 06$) between the two sets, indicating that clones in multi-leader clone sets tend to be longer than those in single-leader sets.

Similarly, we examined the sizes of single- and multi-leader clone sets, as illustrated in Fig.6. The average sizes of the single- and multi-leader clone sets were found to be 2.23 and

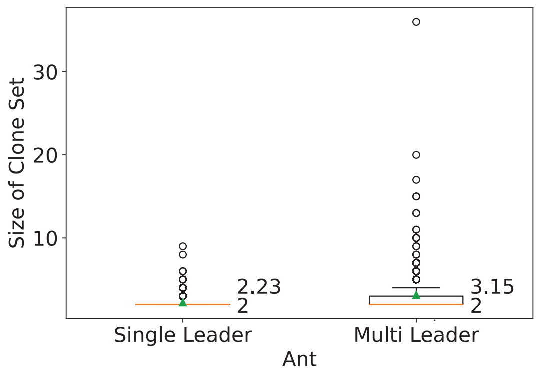


Fig. 6. Size of Single-and Multi-Leader Clone Sets in Ant

3.15, respectively, with both having median values of 2. The Mann-Whitney U-test showed a significant difference ($p = 4.89E-17$) between the two sets, indicating that multi-leader clone sets tend to be larger in size than single-leader sets.

(All Projects) We have extended the analysis to all projects. The ratio of single-leader clone sets to all clone sets for each project ranges from 17.3% to 99.2%, with both an average and median of 66.7%. This means that an average of 33.3% of clone sets are multi-leader and composed of different leader authors. We had expected that the ratio of the multi-leader clone sets to all clone sets, ranging from 0.8% to 82.7%, would correlate to the size of the project (total lines of code) or the total number of authors. However, we found no correlation between them. This means that an increase in total lines or committed authors does not necessarily lead to an increase in multi-leader clone sets.

We have examined the difference in clone lengths between single- and multi-leader clone sets for all projects, as we did for the Ant project. The Mann-Whitney U-test was performed independently for each project to determine if there was a significant difference in clone lengths. Our analysis confirmed that out of all 153 projects, 34 had longer multi-leader clone sets than single-leader sets. For the remaining 119 projects, which comprise the majority, there was no significant difference in the lengths of the single- and multi-leader clone sets.

We performed the Mann-Whitney U-test for each project to determine if there was a significant difference in the clone set sizes between single- and multi-leader sets. Our analysis revealed that 105 out of 153 projects (68.6%) showed larger sizes of the multi-leader clone sets with significant differences. This means that in two-thirds of the projects, the sizes of the multi-leader clone sets, which is the number of instances of each clone set, are larger than those of the single-leader sets.

Answer to RQ2: For all projects, an average of one-third (33.3%) of the clone sets are contributed by multiple leaders, while two-thirds (66.7%) are contributed by a single leader. An increase in total lines or committed authors does not necessarily lead to an increase in multi-leader clone sets. In approximately two-thirds (68.6%) of all projects, multi-leader clone sets have larger clone set sizes compared to single-leader ones.

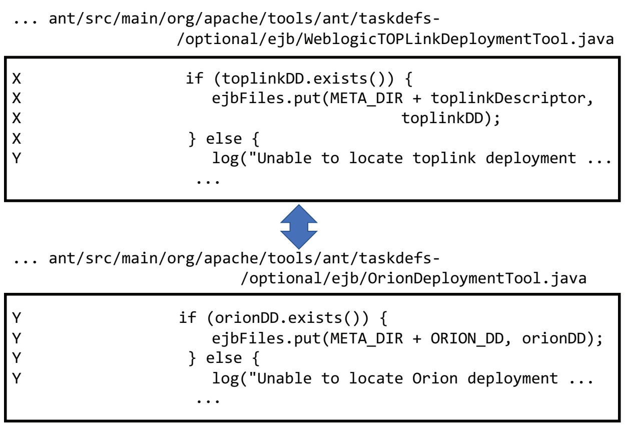
VI. Discussions

A. Contribution to Clone Lines (RQ1)

The answer to RQ1 states that authors who contributed non-clone lines also contributed clone lines. This result aligns very well with our natural intuition for code clone authors and it is statistically validated with 98% projects (150 out of 153 projects). The outlier projects had authors who created mainly code clones but no other code. This might happen if there are developers contributing only to maintenance or refactoring.

B. Different Authors in a Clone Set (RQ2)

The answer to RQ2 states that two-thirds of clone sets are mainly contributed by single leading authors, while the remaining one-third are mainly contributed by multiple leading



```
... ant/src/main/org/apache/tools/ant/taskdefs-
                    /optional/ejb/WeblogicTOPLinkDeploymentTool.java

X                 if (toplinkDD.exists()) {
X                    ejbFiles.put(META_DIR + toplinkDescriptor,
X                                         toplinkDD);
X                 } else {
Y                    log("Unable to locate toplink deployment ...
                  ...

... ant/src/main/org/apache/tools/ant/taskdefs-
                          /optional/ejb/OrionDeploymentTool.java

Y                 if (orionDD.exists()) {
Y                    ejbFiles.put(META_DIR + ORION_DD, orionDD);
Y                 } else {
Y                    log("Unable to locate Orion deployment ...
                  ...
```

Fig. 7. An Example of Multi-Leader Clone Set

authors. Fig.7 depicts a case of the multiple leading authors found in the Ant project. In this example, a type-2 clone pair is presented, with the left column showing the authors of the lines in the right column. By the analysis of timestamps, the upper snippet, which was mainly contributed by author X, is earlier than that of the lower snippet contributed by author Y. It appears that the upper snippet or another clone snippet was copied and pasted to create the lower one.

It is important to recognize a case such that an author A creates a code snippet S_A earlier and a different author B later copies and makes a snippet S_B so that a code clone pair S_A, S_B is formed. In this case, author A might modify S_A to a new snippet S'_A without knowing the existence of S_B, and inconsistency between S'_A and S_B would easily happen. We will discuss this issue later in VI-D.

C. Multi-Leader Clone Set (RQ2)

As stated in the answer to RQ2, we employed a method called single- and multi-leader to measure the major contributing authors of a clone set. This method helps to understand the majority of the clone's contributors, suppressing small contributor effects. We also measured the number of clone sets committed by a *purely single author*, without using the concept of the leader, and calculated the ratio of purely single-author clone sets to all clone sets. Our results showed that the average ratio of the purely single-author clone set for all projects was 34.4%, which is far less than the 66.7% of the single-leader clone set ratio. This indicates the presence of minor contributions from non-leading contributors that have affected the ratio.

In a previous study, the ratio of purely single-author clone sets (classes) was reported as 66.3% [8]. This value is higher than our purely single-author clone set average of 34.4%, which can be attributed to the fact that the previous study is an analysis result of a closed development project with limited members, whereas our study analyzed open development projects with the participation of many developers.

D. Implication for Code Clone Management System

Tab.I shows that approximately 18.5% of the source code is composed of code clones, which is consistent with previous research findings [9]. RQ1 suggests that no special but ordinary developers or maintainers would easily create, delete, or change code clones during their daily activities. These facts indicate that code clones have a fairly large presence in the source code. As code clones establish logical dependencies between code snippets, special attention must be given to them. Previous research has addressed this topic, suggesting features to identify, track, and report code clones and their changes, with a focus on possible bug-inducing activities [5], [17], [22]. Therefore, we consider that source code management tools must have features that can effectively search, trace, and report these logical dependencies generated by code clones.

In addition, a monitoring feature of the clone's consistency, which continuously watches addition, deletion, and changes to all source code, and identifies changes in code clone sets, is important. As mentioned in Sec.VI-B, the author of the original snippet might not know the existence of the clones created by other authors, the continuous monitoring by tools would help to happen inconsistency between logically aligned code snippets.

Based on RQ2, one-third of code clone sets are contributed by multiple leaders, i.e., multiple authors. It is intuitive to assume that multiple-author code snippets require more attention than single-author snippets because inconsistent editing may easily occur due to the involvement of multiple individuals. While validating the impact on the code quality of single versus multiple authors on code snippets would be an interesting future research topic, a bug prediction system that incorporates clone information associated with their authorship could expedite software maintenance tasks.

VII. Threats to Validity

A. Internal Validity

We have used a clone detector CCFinderSW in our study. As reported by other researchers [3], [19], different code clone detectors may report clones differently for the same target, which could potentially affect our results. Although the accuracy of CCFinderSW has been confirmed by its developer [20], it would be worthwhile to explore the use of other detectors such as NiCad, CCFinderX, or SourcererCC to strengthen our result. NiCad or SourcererCC, for example, are capable of detecting type-3 clones directly, which could expand our analysis to include type-3 clones, even though many parts of those could be detected and included as type 1 or 2 clones by our approach. We have primarily used the default parameters of CCFinderSW, but altering them could affect the analysis results. Investigating how the results vary with different combinations of parameters is an avenue for future work.

The target projects may include machine-generated code in their code base and it might affect our result. This would be mitigated by introducing an automatic identification method for the machine-generated code [21].

We have relied on the author's information provided by Git for this analysis. However, Git's author information may not always be reliable [4]. For example, if the original author used multiple names, they may be recognized as different individuals. Additionally, author information might be overridden by Git commands by intention or accidentally. To mitigate these risks, we would employ approaches to identify and merge the author information with email addresses or other associated information [1].

B. External Validity

In this research, we analyzed 153 repositories from Apache projects, which might be a smaller size sample compared to the gigantic real-world OSS projects. However, our sample includes repositories of various sizes with a range of clones, as discussed in Sec.IV-C and V. As such, we believe that our findings could be applicable to other OSS projects or software products, although further research would be needed to validate this. Our analysis focused only on Java programs in Java-oriented projects. Different programming languages with different project structures may exhibit distinct characteristics in terms of code clones. Exploring languages and project structures for the characteristics of clones would be an interesting direction for future research.

VIII. Conclusion

We have presented our analysis of 153 Apache projects aimed at characterizing the authors of code clones within these projects. our validation has revealed a linear correlation between the contributions of non-clone lines and clone lines, meaning that authors who contribute to many non-clone lines also tend to contribute to many clone lines. In addition, we have discovered that two-thirds of clone sets are contributed by single leaders, while the remaining are contributed by multiple leaders which might increase potential inconsistency risks. These results confirmed our intuitive understanding of code clone characteristics with empirical evidence, and we have explored the implications of developing an effective clone management tool.

In our future work, we plan to continue our empirical investigation of clone authorship in relation to code quality and faults. Previous research has shown that clones with multiple authors are more frequently changed [8]. However, the analysis in their study was limited to only one project, and no relation to faults was shown. Therefore, we aim to extend our analysis to multiple target projects and specifically investigate the relationship between clone authorship and the occurrence of faults. Additionally, a detailed analysis of clone authorship, including non-leading authors excluded in this study and authors who disappeared during code evolution, would be an interesting topic for future study.

Acknowledgment

This work is supported by JSPS Grants-in-Aid for Scientific Research, Category (B), 23H03375, and Nanzan University Pache Research Subsidy I-A-2 for the 2023 academic year. We are grateful to Ayu Kojima and Yuka Sato for their contributions to the clone data analysis.

References

[1] S. Amreen, A. Mockus, R. Zaretzki, C. Bogart, and Y. Zhang, "ALFAA: active learning fingerprint based anti-aliasing for correcting developer identity errors in version control systems," *Empir. Softw. Eng.*, vol. 25, no. 2, pp. 1136–1167, 2020.

[2] L. Barbour, F. Khomh, and Y. Zou, "An empirical study of faults in late propagation clone genealogies," *J. Softw. Evol. Process.*, vol. 25, no. 11, pp. 1139–1165, 2013.

[3] S. Bellon, R. Koschke, G. Antoniol, J. Krinke, and E. Merlo, "Comparison and evaluation of clone detection tools," *IEEE Transactions on Software Engineering*, vol. 33, no. 9, pp. 577–591, 2007.

[4] C. Bird, P. C. Rigby, E. T. Barr, D. J. Hamilton, D. M. German, and P. Devanbu, "The promises and perils of mining git," in *6th Int. Working Conf. Mining Software Repositories*, 2009, pp. 1–10.

[5] E. Duala-Ekoko and M. P. Robillard, "Tracking code clones in evolving software," in *29th ICSE*, 2007, pp. 158–167.

[6] git-scm.com, "Show what revision and author last modified each line of a file," *git–blame Manual Version 2.40.0*, 06 2022.

[7] A. Goon, Y. Wu, M. Matsushita, and K. Inoue, "Evolution of code clone ratios throughout development history of open-source c and c++ programs," *Int. Workshop on Software Clones (IWSC)*, pp. 47–50, 2017.

[8] J. Harder, "Code clone authorship — a first look," *Softwaretechnik Trends*, vol. 32, no. 2, pp. 25–26, 2012.

[9] A. Honda, H. Aman, T. Sasaki, and M. Kawahira, "Investigation of convergence tendency of code clone ratio in open source development," *IEICE Transactions D*, vol. J97-D, no. 7, pp. 1213–1215, 2014.

[10] H. Hu, H. Zhang, J. Xuan, and W. Sun, "Effective bug triage based on historical bug-fix information," in *2014 IEEE 25th International Symposium on Software Reliability Engineering*, 2014, pp. 122–132.

[11] K. Inoue and C. K. Roy, Eds., *Code Clone Analysis, Research, Tools and Practices*. Springer, 2021.

[12] V. Kalgutkar, R. Kaur, H. Gonzalez, N. Stakhanova, and A. Matyukhina, "Code authorship attribution: Methods and challenges," *ACM Comput. Surv.*, vol. 52, no. 1, feb 2019.

[13] M. Kim, V. Sazawal, D. Notkin, and G. Murphy, "An empirical study of code clone genealogies," *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 5, pp. 187–196, 2005.

[14] J. Krinke, N. Gold, Y. Jia, and D. Binkley, "Distinguishing copies from originals in software clones," in *Proc. 4th International Workshop on Software Clones*, ser. IWSC '10. ACM, 2010, p. 41–48.

[15] Z. Li, G. Q. Chen, C. Chen, Y. Zou, and S. Xu, "Ropgen: Towards robust code authorship attribution via automatic coding style transformation," in *44th ICSE*, ser. ICSE '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1906–1918.

[16] M. Linares-Vásquez, K. Hossen, H. Dang, H. Kagdi, M. Gethers, and D. Poshyvanyk, "Triaging incoming change requests: Bug or commit history, or code authorship?" in *28th ICSM*, 2012, pp. 451–460.

[17] M. Mondal, C. K. Roy, B. Roy, and K. A. Schneider, "Fleccs: A technique for suggesting fragment-level similar co-change candidates," in *29th Int. Conf. Program Comprehension (ICPC)*, 2021, pp. 160–171.

[18] T. Moriwaki, H. Igaki, Y. Yamanaka, N. Yoshida, S. Kusumoto, and K. Inoue, "Towards an analysis of who creates clone and who reuses it," in *8th Int. Workshop on Software Clones (IWSC)*, 2014, pp. 1–5.

[19] C. K. Roy, J. R. Cordy, and R. Koschke, "Comparison and evaluation of code clone detection techniques and tools: A qualitative approach," *Science of Computer Programming*, vol. 74, no. 7, pp. 470 – 495, 2009.

[20] Y. Semura, N. Yoshida, E. Choi, and K. Inoue, "Ccfindersw: Clone detection tool with flexible multilingual tokenization," in *24th Asia-Pacific Software Engineering Conference (APSEC)*, 2017, pp. 654–659.

[21] K. Shimonaka, S. Sumi, Y. Higo, and S. Kusumoto, "Identifying auto-generated code by using machine learning techniques," in *7th Int. W. Empirical Software Engineering in Practice (IWESEP)*, 2016, pp. 18–23.

[22] S. Tokui, N. Yoshida, E. Choi, and K. Inoue, "Clone notifier: Developing and improving the system to notify changes of code clones," in *27th Int. Conf. on Software Analysis, Evolution and Reengineering (SANER)*, 2020, pp. 642–646.

[23] C. Zhang, S. Wang, J. Wu, and Z. Niu, "Authorship identification of source codes," in *Web and Big Data*. Cham: Springer International Publishing, 2017, pp. 282–296.