An Empirical Study on the Usage and Evolution of Identifier Styles in Practice

Jingxuan Zhang*, Weiqin Zou, Zhiqiu Huang
College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, China
{jxzhang, weiqin, zqhuang}@nuaa.edu.cn

***Abstract*—Identifiers play an important role in helping developers comprehend and maintain source code. In practice, developers usually employ two widely-used identifier styles, i.e., *snake case* and *camel case*, to format identifiers to make them understandable and informative. Despite researchers have empirically investigated the impacts of identifier styles on code comprehension activities, the usage and evolution of identifier styles, however, have not been fully explored. How are individual identifier styles formed in practice? How would identifier styles change and evolve? What are the potential impacts of identifier style-changes? Questions like these are important but have not been fully answered yet. In this paper, we conducted an empirical study on 9,792 GitHub projects to gain some insights into these problems. Specifically, we first analyzed how different identifier styles were formed in real software projects. Next, we explored the change patterns of identifier styles along with the project evolution. Finally, we investigated the potential impacts as well as categories of identifier style-changes. Our empirical results achieved some interesting findings. For example, we first reported some identifier style-change patterns (e.g., *snake case* –> *camel case* –> *snake case*), which could help developers resolve style-change problems in practice. Our study also provided some hints for researchers and developers when they use specific identifier styles in programs. For example, when researchers explore the impacts of identifier styles on code comprehension, they are suggested to consider the imbalanced distribution phenomenon of individual identifier styles. Besides, it is worthwhile for developers to build an identifier style-change prediction and propagation tool to reduce the style-change costs.**

***Index Terms*—Source Code Analysis, Identifier Style, Empirical Study, GitHub**

I. Introduction

Source code comprehension is a common and important activity conducted by developers in their software development process [1], [2]. In practice, developers often resort to source code lexicon, which provides the primary meaning of source code concepts, to understand a piece of code [3]. Among the various types of source code lexicon, identifiers (which take up about 70% of code lexicon) play a fundamental role in expressing the literal meaning of source code [4], [5]. Despite developers have considerable lexical and syntactic freedom in defining identifiers, it is generally believed that poor identifiers set up barriers to source code comprehension and even increase fault-proneness for projects [6]. Hence, developers are encouraged to define meaningful identifiers [7].

To construct readable and informative identifiers, developers usually combine multiple terms together, including dictionary words, abbreviations, and acronyms [8], [9]. There are two typical combination styles (aka. identifier styles in this paper) for integrating multiple terms as identifiers, namely the *snake case* style and the *camel case* style [10], [11]. The *snake case* style is the practice of writing identifiers in which terms are separated by one or several underscores ("_") and the initial letter of each term starts with either case, e.g., *table_editor* or *table_Editor*; while the *camel case* style is the practice of defining identifiers in which the separation of terms are only indicated with a single capitalized initial letter and the first term starts with either case, e.g., *urlParser* or *UrlParser*. Identifiers neither with the *snake case* style nor the *camel case* style are put into the *other* style in this study, e.g., *argvalue*.

In the literature, researchers have investigated the impact of the *snake case* and *camel case* identifier styles on source code reading and comprehension [6], [10], [12]. However, there is still limited empirical evidence about the practical usage of identifier styles and the changes made to identifier styles during the project evolution. How are individual identifier styles formed in real software projects? Are there any evolution patterns for identifier styles and what are the potential impacts of identifier style-changes? Questions like these are important but have not been fully answered by researchers yet [13].

To gain some insights into these problems, we conducted an empirical study to explore the usage and evolution of identifier styles. We first explored how different identifier styles were formed, especially for those identifiers with the *other* style. Then, we explored how identifier styles evolved by mining identifier style-change patterns. Finally, we investigated the potential impacts of identifier style-changes as well as their categories. Our analyses were based on 9,792 Java projects hosted in GitHub. For each project, we extracted identifiers and obtained their corresponding styles. We also extracted the code commit history for identifiers. Then, we applied some mathematical statistical methods with some source code analysis tools to these collected data to study the usage and evolution of identifier styles in practice.

Through experiments, we achieved some interesting results. Specifically, by exploring the construction of identifiers with different styles, we find that identifiers with the *snake case* style are largely (63.40%) constructed in the *partly separated snake case* form; most (87.62%) of identifiers with the *camel*

* Corresponding Author: Jingxuan Zhang.

case style are defined in the *lower camel case* style. By analyzing those identifiers with the *other* style, we find that most of them are dictionary words (45.40%), abbreviations and acronyms (19.80%), and characters combined with digits (14.40%).

As related to identifier style evolution, we observe that more than 75% of identifiers' styles change only once during the project evolution, among which a large portion (59.03%) of style-changes happened between the *camel case* and *other* styles. For identifiers with multiple style-changes, the top 6 common style-change patterns only involve two changes between arbitrary two styles. Interestingly, many identifiers with the initial *camel case* (or *snake case*) style would finally return to the original style after experiencing several times in style changing (e.g., *camel case* –> *snake case* –> *other* –> *snake case* –> *camel case*). In addition, even though style-changed identifiers involve 6.52% of source files on average, they tend to be exposed or used by other parts of the projects with *non-private* access modifiers and have a wide range of reference times. Besides, 52.34% of style-changes are *complex* with more than one term modified, the style-changes are made mainly to narrow down the semantic meanings of identifiers.

The major contributions of this paper are summarized as follows:

- We perform in-depth analyses on the usage and evolution of identifier styles on 9,792 Java projects in GitHub. The source code and dataset to reproduce the results in this paper are made publicly available[1].
- We investigate several key questions related to identifier styles, including how individual identifier styles are formed, their evolution patterns, the potential impacts, and style-change categories.
- Our experimental findings provide some hints and implications for developers and researchers when they resolve identifier-style related problems.

II. Research Questions

In this paper, we aim to explore the usage and evolution of identifier styles in practice. In order to achieve this goal, we study the following Research Questions (RQs):

RQ1: How are identifiers with different styles constructed?

Identifiers with the same style can be constructed in different forms. Understanding the concrete constructions of identifiers with different styles could help us better understand developers' practice in using specific identifier styles, which could further work as the analyses basis for solving identifier-related problems [14], [15].

RQ2: How does identifier style change along with the project evolution?

The style of an identifier may change along with the project evolution [16], [17]. Up to now, no one has reported and summarized the evolution characteristics of identifier styles in practice. Hence, in order to fill this gap, we investigate the evolution characteristics and patterns of identifiers in this RQs. Understanding the evolution characteristics of identifier styles could help to reduce the costs and risks that are associated with identifier style modifications [18], [19].

RQ3: What are the potential impacts of identifier style-changes?

After identifying style-change patterns, we further investigate the potential impacts of these style-changes. Identifier style-changes can be viewed as a type of identifier renaming, which are associated with costs and risks [3]. Understanding the potential impacts of identifier style-changes could help to reduce possible manual efforts and relevant costs as well as risks.

RQ4: What are the categories of identifier style-changes?

After figuring out the identifier style-change patterns and their potential impacts, we proceed to explore the categories and percentages of identifier style-changes from different dimensions. Understanding the style-change categories could help developers have an in-depth understanding on when and how to perform style-changes and make better evaluations to the impacts of style-changes for identifiers. In addition, we can also find some clues to the potential reasons of why developers change styles of identifiers from this RQ.

III. Experimental Setup

In this section, we first describe target projects for experiments and the basic data exploration. Then, we present the design of four RQs.

A. Target Projects

Our experimental projects are from an existing dataset provided by Allamanis and Sutton [20]. This dataset is often used by researchers to explore developers' coding practice [21], [22]. It contains 14,785 Java projects (each has at least one fork in GitHub), with many popular projects also being included, such as ElasticSearch[2] and Apache Maven[3]. These projects are of different sizes and come from different domains, such as framework, library, and database. Such a variety in project scales and domains makes it suitable for our analyses of identifier styles in this paper.

For each project, we downloaded its latest version (to 1st February 2021) so that we could obtain the exact metric values and sufficient historical data from each project to support our analyses. Then, we employed the Java Parser tool[4] to parse each source file within a project. The Java Parser tool could check the syntax correctness of a source file by transforming it into an Abstract Syntax Tree (AST) [23]. In our study, if a project contained source files that had syntax errors (i.e., syntax errors in the code itself) reported by the Java Parser tool, it would be removed from the dataset. This strategy could help us avoid potential problems caused by the biased project data. With this strategy, 1,284 projects were removed from the dataset. Besides, we also filtered out the projects with less

[1]https://github.com/APIDocEnrich/IdentifierStyleAnalyses

[2]https://github.com/elastic/elasticsearch

[3]https://github.com/apache/maven

[4]https://javaparser.org/

TABLE I
SUMMARY OF CHARACTERISTICS ON 9,792 GITHUB JAVA PROJECTS.

Characteristic	Min	Median	Max	Mean	St. Dev.
Watcher	0	4	3,500	21.43	89.15
Star	0	8	50,900	150.81	1,180.74
Fork	1	5	26,700	69.90	543.44
Contributor	1	2	1,516	7.19	29.42
Commit	1	65	449,885	701.35	6,165.52
File	1	35	63,238	212.99	1,151.07
LOC	149	3,479	10,060,819	30,318.01	189,475.20

than 100 identifiers in total from the dataset. This could help us remove some toy projects. Finally, 9,792 projects were left as our final experimental projects. Table I shows the basic statistics of these projects.

B. Preliminary Data Exploration

Before investigating RQs, we first perform some preliminary data exploration to better answer these RQs.

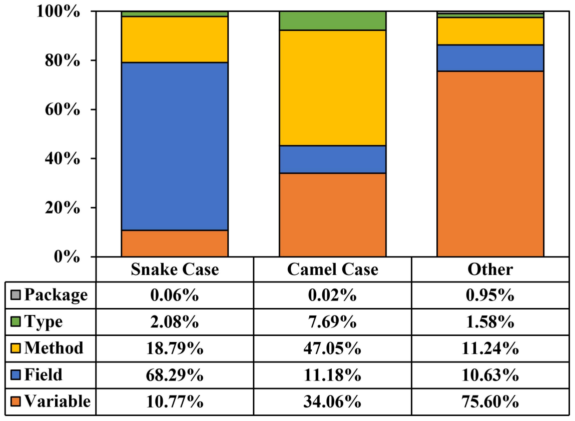
We mainly focus on three identifier styles in this study, namely *snake case*, *camel case*, and *other*. Specifically, if an identifier contained at least one "_", it was treated as with the *snake case* style. If the identifier contained a series of lowercase letters mixed with uppercase letters (may also contain digits) and at least one uppercase letter in the middle, it was regarded as with the *camel case* style. Identifiers neither belonged to the *snake case* nor *camel case* styles were treated as with the *other* style.

According to our statistics, there are a total of more than 57.11 million identifiers (i.e., site of identifier definition) in all 9,792 projects. Identifiers with the *snake case* style only take up 6.38% over all identifiers; while the *camel case* style is the most frequently used style in defining identifiers, with 49.45% of identifiers belonging to this style on the whole. In addition, 44.17% of identifiers are with the *other* style.

We also explored the correlation between different identifier styles and different identifier categories. In this study, we categorized all identifiers into five categories, i.e., package names, type names (including class names, interface names, and enumeration names), method names, field names, and variable names. Within each identifier category, we calculated the ratios of three identifier styles in all the projects. In addition, we also computed the ratios of each identifier category across all identifier styles to better uncover their potential relationships.

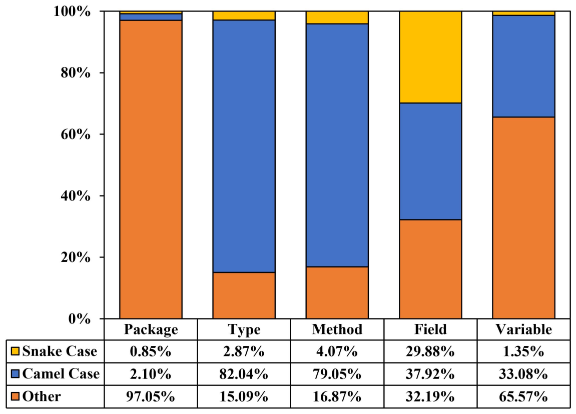
Fig. 1 presents the distributions of the three styles of identifiers over five identifier categories. We can see that most (68.29%) of identifiers with the *snake case* style are field names. A large part of identifiers with the *camel case* style are method names (47.05%) and variable names (34.06%). For identifiers with the *other* style, more than three quarters (75.60%) are variable names.

Similarly, Fig. 2 shows the distributions of different identifier categories over the three identifier styles. We can see that more than 97% of package names are defined following the *other* style. This indicates that the *snake case* and *camel case* styles are generally not applied to package names in practice. As for type names and method names, more than 82% and 79% of them are defined using the *camel case* style. Field names are almost evenly distributed in all three identifier styles, which means it may be hard to choose a proper style for a field name in practice. As for variable names, about two thirds (65.57%) of them are with the *other* style and the rest (33.08%) are with the *camel case* style. This means developers seldom employ the *snake case* style to define variable names.



	Snake Case	Camel Case	Other
▨ Package	0.06%	0.02%	0.95%
🟩 Type	2.08%	7.69%	1.58%
🟨 Method	18.79%	47.05%	11.24%
🟦 Field	68.29%	11.18%	10.63%
🟧 Variable	10.77%	34.06%	75.60%

Fig. 1. The distributions of three identifier styles over five kinds of identifiers.



	Package	Type	Method	Field	Variable
🟨 Snake Case	0.85%	2.87%	4.07%	29.88%	1.35%
🟦 Camel Case	2.10%	82.04%	79.05%	37.92%	33.08%
🟧 Other	97.05%	15.09%	16.87%	32.19%	65.57%

Fig. 2. The distributions of five kinds of identifiers over three identifier styles.

C. Design of RQs

RQ1: How are identifiers with different styles constructed?

As mentioned above, there are more than 57.11 million identifiers in total in the experimental projects. It is impossible for us to manually explore the construction of all the identifiers. Hence, we sample some identifiers to investigate this RQ. Specifically, we first sampled 500 identifiers for each identifier style from the target projects. We employed the stratified random sampling method (a sampling technique widely used by researchers when trying to draw conclusions from different data categories [24]) to obtain those to-be-analyzed identifiers. In the stratified random sampling method, we randomly sampled identifiers from each category of which the percentage was equal to the proportion of this category as a whole for a specific identifier style. Since the process of stratifying reduced sampling errors and ensured a great level of representation [24], the sampled identifiers could reflect the diversity of the whole population. Next, two authors of this study independently analyzed all the sampled identifiers and manually checked the programming context to figure out how these identifiers were constructed. In this step, the two authors could check the results more than one time until they thought that there was no need to change the evaluation results. Finally, the two authors discussed their results together and solved any disagreement to reach a consensus on how identifiers with different styles were constructed. Kappa inter-rater agreement was also computed to measure the consistency degree between the two authors' results [25].

RQ2: How does identifier style change along with the project evolution?

Given a specific identifier, we tracked its style-change history by using the Git *log* command "git log -L <start>, <end> : <file>", where we specified the location (line number) of the identifier as both the *start* as well as *end* parameters and the source file of the identifier as the *file* parameter. This command could output all the commits with detailed changes to the location where the identifier was defined chronologically. By comparing the current version with its historical versions, we could figure out whether the style of this identifier had been changed. If there were any style-changes for this identifier, we generated a style-change chain for it based on the change history. Finally, after analyzing all the style-changed identifiers, we calculated the frequencies of style-change chains and regarded the most frequent style-change chains as style-change patterns for identifiers.

The Git *log* operations on specific identifiers required considerable processing time (usually one to several seconds for an identifier), which was hard to be estimated, especially for those identifiers which had tremendous historical commits. Considering that there were a large number of projects (almost 10 thousand) and much larger number of contained identifiers (more than 57.11 million), rather than analyzing all the projects, we randomly selected 1,000 projects to perform relevant analyses in this RQ.

RQ3: What are the potential impacts of identifier style-changes?

Specifically, we explored the potential impacts of identifier style-changes based on the same 1,000 projects in RQ2 from three aspects. First, for each of the 1,000 projects, we calculated the percentage of source files in which the styles of identifiers had ever been changed. In such a way, we could exactly know how many source files were directly affected.

Second, since different identifier categories had different effect scopes, we further investigated the affected scopes of identifiers whose styles have been changed with associated access modifiers. In the Java programming language, the access levels of types, methods, and fields were controlled by four kinds of access modifiers, i.e., *public*, *protected*, *private*, or no modifier (we call it *default*). Changing the style of a *public* method had a higher cost than that of a *private* method, since it might break backward compatibility and increase integration costs [3]. To measure the affected scopes, we calculated the ratios of the four types of access modifiers in three identifier categories, i.e., types, methods, and fields.

Third, we counted the reference or usage time for each style-changed identifier to evaluate how many modification operations might to be conducted. After developers changed the style for a specific identifier, they should propagate the style-change to other code parts so that the project could still be compilable and runnable. By calculating the reference time for each style-changed identifier, we could know the number of possible modification operations so as to assess the potential impacts of those style-changed identifiers.

RQ4: What are the categories of identifier style-changes?

As a type of identifier renaming, identifier style-changes can be classified into different categories from different dimensions. The detailed and specific style-change categories had been extensively investigated by Arnaoudova et al. [3], who proposed a systematic identifier renaming taxonomy. This taxonomy contained four orthogonal dimensions, i.e., *entity kinds*, *forms of renaming*, *semantic changes*, and *grammar changes*, each of which contained several categories [3]. For example, in the *forms of renaming* dimension, identifier style-changes could be divided into four categories, including *simple*, *complex*, *formatting only*, and *term reordering*.

We investigated the categories and their percentages by manually classifying identifier style-changes based on the proposed taxonomy [3]. Similar to RQ1, we first used the stratified random sampling method to sampled 500 identifiers whose styles had been changed. As some sampled identifiers might have multiple style-changes, for these identifiers, we split their multiple style-changes into several single style-changes. Then, two authors of this paper manually and independently analyzed the styles of those sampled identifiers before and after each single style-change. Each single style-change was placed into different categories of four orthogonal dimensions. Next, the two authors discussed the divergences to reach an agreement about the final categories of identifier style-changes. At last, we calculated the percentage of each category for all the dimensions and reported the agreement degree between the two authors [25].

IV. Experimental Results

In this section, we present the empirical results of each RQ.

RQ1: How are identifiers with different styles constructed?

Based on sampled identifiers, we find that developers generally employ "_" to combine terms to define identifiers with the *snake case* style in two forms. In the first form, each constitutive term is separated by "_"; we call this form as *fully separated snake case* in this study. For example, the identifier *sample_data_type* is in accordance with this form. The ratio of identifiers with this form is 36.60%. In the second form, only one or several key positions in identifiers are embedded with "_"; we call this form as *partly separated snake case*. For example, the identifier *exitbutton_onclick* is in line with this form. Identifiers with the second form take up 63.40%. This indicates that developers tend to use *partly separated snake case* to define identifiers with the *snake case* style.

For the *camel case* style, there are generally two types, i.e., the *upper camel case* and the *lower camel case*. In the *upper camel case*, the first letters of all constitutive terms are capitalized; while in the *lower camel case*, the first letter is in lowercase, but the first letters of the subsequent terms are in uppercase. During our analyses, we find that identifiers with the *lower camel case* style take up 87.62% and the rest are identifiers with the *upper camel case* style. This means developers are willing to use the *lower camel case* style to define identifiers with the *camel case* style.

We also investigate the distributions of the two types of the *camel case* style in different identifier categories. We find that about 50% of identifiers with the *lower camel case* style are method names. This phenomenon is consistent with the Java code conventions to some extent, which state that *"method names usually start with a lowercase letter with the first letter of each internal word capitalized"*. As for the *upper camel case* style, 72.80% of identifiers with this style are type names and field names. This result is also in line with the Java code conventions, which state that *"type names should be in mixed case with the first letter capitalized"*. These results indicate that developers often follow the code conventions related to the *camel case* style when they define identifiers. This may be due to that some IDEs like Eclipse could remind developers of these code conventions when they program.

In terms of the *other* style, we manually identify five forms that developers follow to define identifiers with this style:

1) Some identifiers are dictionary words that are commonly seen in programs, e.g., *value* and *name*. Identifiers with this formation take up 45.40% in all sampled identifiers.
2) Some identifiers are abbreviations and acronyms. For example, *arg* (usually standing for *argument*) and *tmp* (usually standing for *temporary*) are common abbreviations and *XML* (usually standing for *eXtensible Markup Language*) is a common acronym. Identifiers created by this formation take up 19.80% in all sampled identifiers.
3) Some identifiers are composed of characters and digits without other special characters, such as *results1* and *key256*. Identifiers made by this formation take up 14.40% in all sampled identifiers.
4) Some identifiers are constructed by all uppercase or lowercase letters, e.g., *lengthcount* and *ENTITYTYPE*. Identifiers created by this formation take up 13.20% in all sampled identifiers.
5) Some identifiers are arbitrarily created by developers possibly only because some characters are close to each other in the keyboard. For example, *az* and *sdf* are two arbitrarily constructed identifiers. Identifiers made by this formation take up 9.80% in all sampled identifiers.

Note that the five formations mentioned above are not completely orthogonal. An identifier with the *other* style may belong to more than one formations. Besides the above five formations, we also find eight identifiers that are misspelled in the sampled identifiers. Among them, three identifiers are spelled incorrectly because the sequence of some characters is not in a correct order. For example, the identifier *laoder* originally should be *loader*. The other five identifiers are misspelled since one or more characters are accidentally changed to other characters adjacent to them in the keyboard. For example, the identifier *attribites* is supposed to be *attributes*. The Kappa inter-rate agreement value between the two authors is 0.535, showing a moderate agreement [25].

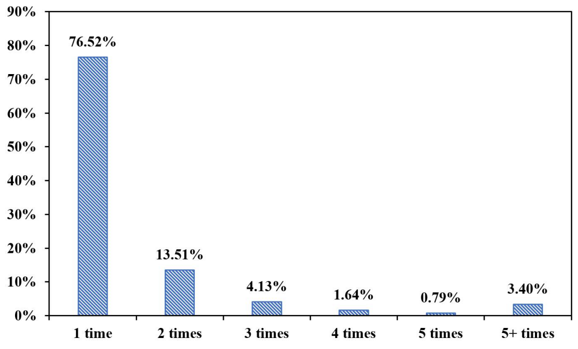


Fig. 3. The distribution of style-change times for identifiers whose styles have ever been changed.

Finding 1: Identifiers with the *snake case* style are largely (63.40%) constructed in the *partly separated snake case* form; most (87.62%) of identifiers with the *camel case* style are defined in the *lower camel case* style; those identifiers with the *other* style are mainly constructed in forms of dictionary words (45.40%), abbreviations and acronyms (19.80%), and characters combined with digits (14.40%).

RQ2: How does identifier style change along with the project evolution?

Identifiers as well as their styles are in constant evolution. According to our statistics, there are totally 20,010 identifiers whose styles have ever been changed during the project evolution, and the percentage of style-changed identifiers with respect to the total number of identifiers in the 1,000 projects is 2.43%. Fig. 3 shows the distribution of style-change times of all style-changed identifiers. From the figure, we can see that the overall change-time trend follows the long-tailed distribution: the percentage of identifiers whose styles changed only once reaches as large as 76.52%; along with the change-time increases, the corresponding percentages rapidly decline. For example, identifiers whose styles have been changed 3 times and 4 times only take up 4.13% and 1.64% respectively. Some identifiers (3.40%) seem to have their styles frequently changed (*5+ times*), with the largest change-times being 9 in our experimental projects. In addition, we observe that there are 10.22% of identifiers whose styles have been changed back to their initial (original) styles after some changes, e.g., *camel case −> snake case −> other −> snake case −> camel case*.

After we obtain the distribution of style-change times, we

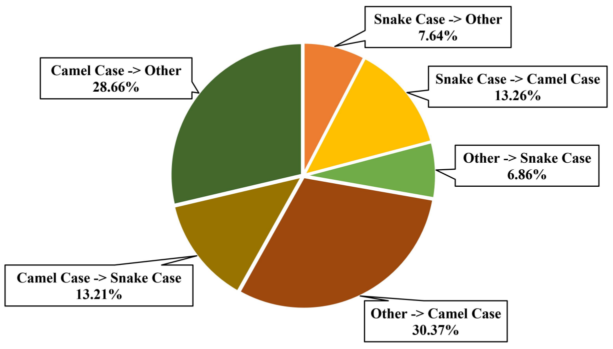


Fig. 4. The distribution of style-change patterns for identifiers whose styles changed only once.

TABLE II
THE TOP 10 IDENTIFIER STYLE-CHANGE PATTERNS.

Rank	Style-Change Patterns
1	Camel Case -> Other -> Camel Case
2	Other -> Camel Case -> Other
3	Snake Case -> Other -> Snake Case
4	Other -> Snake Case -> Other
5	Snake Case -> Camel Case -> Snake Case
6	Camel Case -> Snake Case -> Camel Case
7	Other -> Camel Case -> Snake Case -> Other -> Snake Case
8	Camel Case -> Other -> Snake Case -> Camel Case
9	Camel Case -> Other -> Camel Case -> Snake Case -> Other
10	Other -> Snake Case -> Camel Case -> Other

then study whether the style-changes in individual identifiers would change the overall percentages of three identifier styles in all projects. We find that on the whole, the *camel case* style slightly increases by 1.75%, while the number of identifiers with the *snake case* and *other* styles slightly decrease by 0.82% and 0.93% respectively. This means despite many identifiers may have their styles changed, the overall distribution of the three styles remains relatively steady.

In terms of identifiers whose styles changed only once, Fig. 4 shows all possible one-time style-changes and their corresponding percentages. In the legend of Fig. 4, the styles before and after the arrow (i.e., ->) stand for the styles before and after the change respectively. For example, *camel case* −> *snake case* represents that the original style of an identifier is *camel case*, then it changes to *snake case* during the project evolution. From the figure, we can find that a large number of style-changes (28.66%+30.37%=59.03%) happened between the *other* and *camel case* styles. Relatively, the percentage of style-changes between the *snake case* and *camel case* styles is much lower (13.26%+13.21%=26.47%). In contrast, the style-changes between the *snake case* and *other* styles take up the smallest percentage, i.e., 14.50% in total. Given the style-change statistics among three styles and the previous observation that the number of identifiers with the *camel case* style increases by 1.75%, we can conclude that such an increase is mainly from the *other* style, i.e., identifiers with the *other* style are changed to the *camel case* style.

After we studied the one-time style-changes, we continued to explore style-change patterns of identifiers whose styles changed several times (>= 2). For each of these identifiers, we create a style change chain for it by parsing its style change history. For example, "*camel case* -> *snake case* -> *camel case*" is a style change chain, which means the style of an identifier changed from *camel case* to *snake case*, and then changed back to *camel case* again. After creating change chains for all identifiers with multiple style changes, we accumulate those chains and summarize some common style-change patterns. Table II shows the top 10 common style-change patterns we identified. From the table, we can observe that, each of the top 6 style-change patterns only involves two changes between two identifier styles. For example, the most common style-change pattern (i.e., the top 1 in the table) of identifiers with multiple style-changes is the one that generally changes the style of an identifier from *camel case* to *other*, and finally back to *camel case* again. The rest four style-change patterns (i.e., the 7th to the 10th in the table) mainly involve multiple changes among all three identifier styles. For example, the 8th style-change pattern means the identifier style changes from *camel case* to *other*, then to *snake case*, and finally changes to *camel case* again.

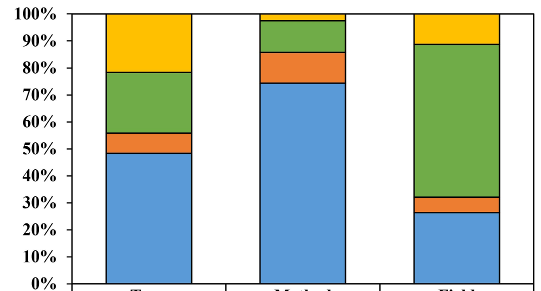
Finding 2: 76.52% of identifiers' styles changed only once during the project evolution, among which 59.03% of changes happened between the *camel case* and *other* styles. For identifiers with multiple style-changes, the top 6 common style-change patterns only involve two changes between arbitrary two styles.

RQ3: What are the potential impacts of identifier style-changes?

By analyzing the sampled 1,000 projects, we find that the average number of source files with identifier styles changed is 12.68; the average percentage of these source files is 6.52%. This indicates that the style-changed identifiers would have an unignorable impact on a few of source files.

When focusing on types, methods, and fields, whose scopes can be controlled by different access modifiers, we show their distributions on the four types of access modifiers in Fig. 5. As for style-changed types, we can see that 48.33% of them are *public*, which can be referred in anywhere in other source files of the same project. In addition, 21.67% and 7.50% of them are *default* and *protected* respectively, which can be accessed within limited scopes (e.g., within the same package). This means that almost 80% of style-changed types can be referred by other source files (with *non-private* access modifiers). In terms of style-changed methods, we can find that 74.35% of them are declared as *public*; and 11.39% and 2.48% are with *protected* and *default* access modifiers respectively. This also indicates that the potential affected scopes could be large for style-changed methods. Unlike types and methods, the changes to field styles have a relatively smaller impact, since 56.61% of fields with style-changes are declared as *private*.

Fig. 6 shows the boxplot of the reference or usage times of identifiers whose styles have been changed separated by



	Type	Method	Field
🟨 Default	21.67%	2.48%	11.28%
🟩 Private	22.50%	11.77%	56.61%
🟧 Protected	7.50%	11.39%	5.71%
🟦 Public	48.33%	74.35%	26.40%

Fig. 5. The distribution of types, methods, fields over four access modifiers.

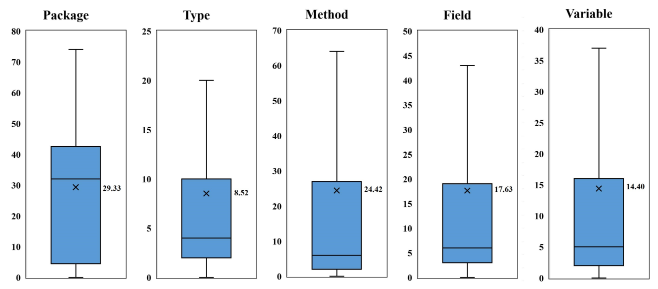


Fig. 6. The reference or usage times of identifiers whose styles have been changed (The average values are shown with × in this figure).

different categories. We can find that the reference times of all the identifier categories are distributed in relative large ranges. Specifically, the reference times of 25% quartile to 75% quartile of packages range from 5 to 42, and the average reference time is 29.33. It shows that once the style of a package is changed, more than 29 source files should be modified to adapt to the style-change on average. In terms of types, the reference times of 25% quartile to 75% quartile range from 2 to 10 with an average value of 8.52. It means that on average more than 8 modification operations should be conducted once the style of a type is changed. As for methods, 25% quartile and 75% quartile of reference times are 2 and 27 with an average value of 24.42. Fields show similar results as methods. The average reference time of fields is 17.63, showing that more than 17 modifications should be operated on average if the style of a field is changed. Since variables can only be used within their corresponding enclosed methods, their reference times are relatively small, ranging from 2 to 16 in terms of 25% quartile and 75% quartile.

Finding 3: Style-changed identifiers directly involve 6.52% of source files on average. These identifiers tend to be exposed to other code parts with *non-private* access modifiers and have a wide range of reference times, which means that substantial modification operations need to be conducted when propagating the style-changes.

TABLE III
THE CATEGORIES AND RATIOS OF IDENTIFIER STYLE-CHANGES.

Dimension	Category	Percentage (%)
Entity Kinds	Package	0.20
	Type	1.20
	Method	28.40
	Field	26.00
	Variable	44.20
Forms of Renaming	Simple	28.72
	Complex	52.34
	Formatting Only	18.09
	Term Reordering	0.85
Semantic Changes	Preserve Meaning	16.21
	Change in Meaning	13.68
	Narrow Meaning	27.16
	Broaden Meaning	13.47
	Add Meaning	6.74
	Remove Meaning	3.58
	None	19.16
Grammar Changes	Part of Speech Change	4.04
	None	95.96

RQ4: What are the categories of identifier style-changes?
We obtained 655 single style-changes for the sampled 500 identifiers after splitting multiple style-changes. Two authors manually classify each single style-change into different categories in different dimensions, and the Kappa inter-rate agreement value is 0.656, indicating a good agreement between two authors [25]. Table III shows the identifier style-change categories and their percentages. From the *entity kinds* dimension, we find that variables take up the largest percentage (44.20%) in all style-changed identifiers. It means that variable styles are more frequently to be changed during the project evolution. The percentages of style-changed methods and fields are similar (28.40% and 26.00%). While the ratios for packages and types are only 0.20% and 1.20% respectively, showing that their styles are seldom changed.

From the *forms of renaming* dimension, we observe that more than a half (52.34%) of style-changes are *complex*, with more than one terms changed. For example, *selection* −> *selectedItem* changes the style from *other* to *camel case* with two terms modified. In contrast, 28.72% of style-changes are *simple*, with only one term changed, e.g., *itemsExpected* −> *items* changes the style from *camel case* to *other* with one term *Expected* deleted. In addition, 18.09% of style-changes are *formatting only*, meaning that only letter cases or term separators have been changed. For instance, *RoutingKey* −> *ROUTING_KEY* makes the style changed from *camel case* to *snake case* by revising its format. Only 0.85% of style-changes exchange the positions of the composing terms, i.e., belong to the *term reordering* category. For example, *tool_select_rectangle* −> *selectRectangleTool* changes the style from *snake case* to *camel case* and reorders the positions of the composing terms.

As an important dimension, the *semantic changes* dimension includes 7 categories. We observe that identifier style-changes belonging to the *narrow meaning* category take up the largest percentage (27.16%). Identifiers with this category change their styles by changing the composing terms into their hyponyms or adding specific terms, e.g., *cache* −> *htmlCache* changes the style from *other* to *camel case* by adding a specific term *html* to narrow down its meaning. The second largest percentage comes from the *none* category, which implies no semantic change and mainly comes from the *formatting only* category in the *forms of renaming* dimension. 16.21% of style-changes belong to the *preserve meaning* category, which preserves the semantic meanings of identifiers by using synonyms and abbreviation expansion. For example, *ddmenu −> dropDownMenu* changes the style from *other* to *camel case* by expanding *dd* to *dropdown*. Style-changes of the *change in meaning* and *broaden meaning* categories take up 13.68% and 13.47% respectively. Here, *change in meaning* means changing the meaning of identifiers; while *broaden meaning* means generalizing the meaning of identifiers. For example, *correlateUnique −> uncorrelate* belonging to the *change in meaning* category changes identifier style from *camel case* to *other* and also changes its meaning to the opposite. In contrast, much fewer style-changes belong to the *add meaning* and *remove meaning*; these style-changes mainly aim to add and remove semantic meanings respectively.

From the *grammar changes* dimension, we discover that 95.96% of style-changes belong to the category of *none*. Only 4.04% of style-changes belong to the *part of speech change* category. For example, *connect −> waitForConnection* changes the identifier style from *other* to *camel case*, in which the verb *connect* is changed to a noun *connection*.

Finding 4: The styles of variables are changed much more frequently than the other identifier categories. 52.34% of style-changes are *complex* with more than one term modified. Many style-changes are made to either narrow down, preserve the identifier meaning or just change the formats of identifiers. 95.96% of style-changes will not change the part of speech of the composing terms.

V. Discussion

In this section, we discuss implications and threats.

A. Implications

Research studies on exploring the impacts of identifier styles on code comprehension need to consider the imbalanced distribution phenomenon of individual identifier styles. According to our preliminary data statistics, the distribution of the *camel case* and *snake case* styles is extremely imbalanced (49.45% vs 6.38%) in 9,792 projects. To the best of our knowledge, existing studies generally evenly sampled identifiers with these two styles when studying their impacts on code comprehension tasks. This is problematic as ignoring the imbalanced style-distribution phenomenon may introduce some biases into the arrived conclusions. Hence, we would suggest researchers and practitioners who attempt to analyze how identifier styles would exactly affect code comprehension activities, to consider the imbalanced distribution problem of identifier styles, e.g., assign different weights to different identifier styles based on the statistics in this study.

Extra efforts need to be spared to eliminate arbitrarily constructed identifiers with the *other* style. We find that a large part (44.17%) of identifiers are with the *other* style (i.e., neither with *camel case* nor *snake case*). Furthermore, among these identifiers, there are as large as 9.80% of identifiers which are constructed arbitrarily without having any meaning or intention (findings of RQ1). In addition, some of them are also misspelled. These poor identifiers would prevent developers from understanding code and may increase project fault-proneness [6]. Hence, it would be beneficial to spare extra efforts to particularly improve the quality of identifiers with the *other* style, e.g., automatically remind or warn developers if they arbitrarily construct identifiers with the *other* style and further provide them with suggested meaningful and informative identifiers.

It is worthwhile to build a style-change prediction model and further explore the reasons behind style-changes. In RQ2, we find that identifier styles would change along with the project evolution. The changes towards identifier styles usually come at costs, e.g., waste developers' time or introduce potential inconsistency problems [3]. Thus, it would be valuable to build a model to precisely predict possible style-changes and recommend the final styles for identifiers. Additionally, despite we discover some interesting style-change patterns, we have no clue about why identifier styles change. In RQ4, we classify identifier style-changes into different categories, which may be regarded as coarse-grained reasons. However, we still need to explore the fine-grained style-change reasons, which could reveal some interesting and worth-to-solve research problems. Actually, we have tried to manually analyze a number of commit messages with the hope to find possible style-change reasons but we failed, due to two reasons. First, identifier style-changes may be viewed as a type of refactoring, but developers are found to not frequently state refactoring activities in commit messages in practice [26], [27]. Second, multiple unrelated changes are usually packed in a single commit [28], again making style-change reasons undocumented. In the future, we plan to conduct a survey among developers to understand the concrete reasons for various style-change patterns identified in our study.

New tools are needed to help developers propagate the style-changes of identifiers. When investigating the impacts of identifier style-changes in RQ3, we find that style-changed identifiers have a wide range of reference times and their potential impacts can be large. Nowadays, existing modern IDEs, such as IntelliJ IDEA and Visual Studio, are embedded with convenient style converter plugins, which can help developers convert selected identifiers between different styles. However, these plugins only focus on changing the styles of identifiers themselves without propagating the style-changes. Manually propagating the identifier style-changes is time-consuming and error-prone, and forgetting to change any one of the references to style-changed identifiers may introduce potential bugs [29]. Considering the large impacts of identifier style-changes, it is worthwhile to construct a tool to help developers propagate identifier style-changes.

B. Threats to Validity

Internal Validity: Threats to internal validity are related to the internal factors that could affect the results. In RQ1 and RQ4, two authors of this study explore the constructions of identifiers and style-change categories by conducting manual qualitative analyses. The manual analyses results may be affected by personal subjectivity. To reduce the influence of personal opinions to the results, we take some measures. For example, each result is analyzed by two authors independently. If there is a discrepancy for a specific result, they would discuss it to reach an agreement. In addition, the representativity of the sampled identifiers in RQ1 and RQ4 could also be a potential threat of this study. To reduce such a threat, we employ the effective stratified random sampling method, which would not lean towards a specific bias or prejudice [24].

External Validity: Threats to external validity concern the generalizability of our empirical results. In this study, all analyses are conducted on open-source Java projects hosted in the GitHub community. The achieved conclusions may not be applicable to the projects programmed by other programming languages, to the closed-source projects, or in other open-source-code-hosted platforms (such as BitBucket or SourceForge). For example, the imbalanced distribution phenomenon of identifier styles may be because Java code conventions promote the use of the *camel case* style. This phenomenon may not occur in other programming languages. However, considering that our experimental projects varied in domains and are of different sizes, we believe that our study could still shed some lights in the usage and evolution of identifier styles.

VI. Related Work

In this section, we briefly introduce the closely related work.

A. The Impact of Identifiers on Software Development

As a major part of source code lexicon, identifiers could influence software development [5], [8]. Binkley et al. investigated the impact of the *camel case* and *snake case* styles on code readability [10]. By measuring the speed and accuracy of manipulating programs, they found that developers using the *camel case* style achieved higher accuracy of the program manipulation than those using the *snake case* style. Following this work, Sharif et al. employed an eye tracker to capture the quantitative results from developers when they were involved in the experiment [12]. They obtained different results as Binkley et al. [10], i.e., there was no difference between identifier styles with respect to the accuracy of manipulating programs. In addition, Sharafi et al. investigated the impact of both genders on the source code reading activities with different identifier styles [30]. They observed that there was no significant difference of the accuracy and the efforts of the code reading between male and female subjects when they were provided with different identifier styles.

Except for exploring the impact of identifier styles, researchers also explored the influence of the other aspects of identifiers. Binkley et al. investigated the balance between longer identifier names and limited programmer memory [31]. They found that longer identifier names took more time to process, comprehend, and manipulate. Butler et al. employed 12 identifier naming guidelines to evaluate the quality of identifiers in 8 open source projects [32]. They found that flawed identifiers and code quality issues were significantly associated. Lawrie et al. found that full-word identifier names can lead to better code comprehension than single-letter identifiers and abbreviations as identifiers [33]. Scanniello et al. studied the influence of full-word identifiers and abbreviations as identifiers on fault fixing, and found no difference between the two types of identifiers [34].

None of these studies explore the practical usage and evolution of identifier styles. The results of this study complement existing studies and could inspire the better usage of specific identifier styles for developers.

B. Identifier Characteristics Analyses

Some studies try to figure out what characteristics make a good identifier. Deissenbock and Pizka pointed out that identifiers should be consistent and concise [35]. However, automatically verifying the consistency and conciseness of identifiers was non-trivial. To resolve this problem, Lin et al. proposed a method to ensure the consistency and conciseness of identifiers [36]. Furthermore, the degree of consistency and conciseness of identifiers could be decreased along with the evolution of the projects [37]. Allamanis et al. [38] and Butler et al. [39] pointed out that when developers defined identifiers, they needed to follow the corresponding code conventions. Hence, identifiers also had the characteristic of normativeness. Falleri et al. stated that identifiers should have the characteristic of understandability [40]. They automatically constructed a wordnet-like identifier network to improve the performance of resolving typical research tasks.

Identifier styles can be also viewed as a guide or best-practice that developers should follow when they define identifiers. The findings of this study can help developers improve the quality of identifiers as well as their programs.

VII. Conclusion and Future Work

It has been a hot research topic to analyze source code lexicon, especially identifiers. Even though researchers have explored the influence of identifiers on code comprehension activities, there are still some underlying open issues that have not been fully investigated. In this study, we conduct a large-scale empirical study on the usage and evolution of identifier styles in practice and achieve some interesting findings. Based on these findings, we summarize some implications that inspire future research in resolving identifier style related problems.

In the future, we plan to extend our work in following aspects. First, we plan to explore the usage and evolution of identifier styles for other programming languages, e.g., C++. Second, we plan to conduct a survey to investigate developer's preference to identifier styles and reasons of style-changes of identifiers. Third, we plan to develop some automatic tools (e.g., a warning tool for identifiers with the *other* style) to help developers perform identifier-related activities.

Acknowledgment

This work is supported by the National Natural Science Foundation of China under Grant No. 61902181 and 62002161, the China Postdoctoral Science Foundation under Grant No. 2020M671489, and the CCF-Tencent Open Research Fund under Grant No. RAGR20200106.

References

[1] S. Kim and D. Kim, "Automatic identifier inconsistency detection using code dictionary," *Empirical Software Engineering*, vol. 21, no. 2, pp. 565–604, Apr. 2016.

[2] X. Xia, L. Bao, D. Lo, Z. Xing, A. E. Hassan, and S. Li, "Measuring program comprehension: A large-scale field study with professionals," *IEEE Transactions on Software Engineering*, vol. 44, no. 10, pp. 951–976, Oct. 2018.

[3] V. Arnaoudova, L. M. Eshkevari, M. D. Penta, R. Oliveto, G. Antoniol, and Y. G. Gueheneuc, "Repent: Analyzing the nature of identifier renamings," *IEEE Transactions on Software Engineering*, vol. 40, no. 5, pp. 502–532, 2014.

[4] J. C. Hofmeister, J. Siegmund, and D. V. Holt, "Shorter identifier names take longer to comprehend," *Empirical Software Engineering*, vol. 24, no. 6, pp. 1–27, 2018.

[5] E. Enslen, E. Hill, L. Pollock, and K. Vijay-Shanker, "Mining source code to automatically split identifiers for software analysis," *In Proceedings of the IEEE International Working Conference on Mining Software Repositories (MSR 09)*, pp. 71–80, 2009.

[6] D. Binkley, M. Davis, D. Lawrie, J. I. Maletic, C. Morrell, and B. Sharif, "The impact of identifier style on effort and comprehension," *Empirical Software Engineering*, vol. 18, no. 2, pp. 219–276, Apr. 2013.

[7] D. Lawrie and D. Binkley, "Expanding identifiers to normalize source code vocabulary," *In Proceedings of the IEEE International Conference on Software Maintenance (ICSM 11)*, pp. 113–122, 2011.

[8] S. Butler, M. Wermelinger, Y. Yu, and H. Sharp, "Improving the tokenisation of identifier names," *In Proceedings of the European Conference on Object-oriented Programming (ECOOP 11)*, pp. 130–154, 2011.

[9] A. D. Lucia, M. D. Penta, and R. Oliveto, "Improving source code lexicon via traceability and information retrieval," *IEEE Transactions on Software Engineering*, vol. 37, no. 2, pp. 205–227, 2011.

[10] D. Binkley, M. Davis, D. Lawrie, and C. Morrell, "To camelcase or under_score," *In Proceedings of the 17th IEEE International Conference on Program Comprehension (ICPC 09)*, pp. 158–167, 2009.

[11] S. Butler, M. Wermelinger, Y. Yu, and H. Sharp, "Exploring the influence of identifier names on code quality: An empirical study," *In Proceedings of the 14th European Conference on Software Maintenance and Reengineering (CSMR 10)*, pp. 156–165, 2010.

[12] B. Sharif and J. I. Maletic, "An eye tracking study on camelcase and under_score identifier styles," *In Proceedings of the 18th International Conference on Program Comprehension (ICPC 10)*, pp. 196–205, 2010.

[13] Y. Hayase, Y. Kashima, Y. Manabe, and K. Inoue, "Building domain specific dictionaries of verb-object relation from source code," *In Proceedings of the 15th European Conference on Software Maintenance and Reengineering (CSMR 11)*, pp. 93–100, 2011.

[14] Y. Cao, Y. Zou, Y. Luo, B. Xie, and J. Zhao, "Toward accurate link between code and software documentation," *Science China Information Sciences*, vol. 61, no. 5, pp. 68–82, 2018.

[15] J. Guo, J. Cheng, and J. Cleland-Huang, "Semantically enhanced software traceability using deep learning techniques," *In Proceedings of the 39th International Conference on Software Engineering (ICSE 17)*, pp. 3–14, 2017.

[16] A. Corazza, S. D. Martino, and V. Maggio, "Linsen: An efficient approach to split identifiers and expand abbreviations," *In Proceedings of the IEEE International Conference on Software Maintenance (ICSM 13)*, pp. 233–242, 2013.

[17] D. Lawrie, D. Binkley, and C. Morrell, "Normalizing source code vocabulary," *In Proceedings of the Working Conference on Reverse Engineering (WCRE 10)*, pp. 3–12, 2010.

[18] Y. Jiang, H. Liu, J. Zhu, and L. Zhang, "Automatic and accurate expansion of abbreviations in parameters," *IEEE Transactions on Software Engineering*, vol. 46, no. 7, pp. 732–747, 2020.

[19] E. Hill, Z. P. Fry, H. Boyd, G. Sridhara, Y. Novikova, L. L. Pollock, and K. Vijay-Shanker, "Amap: Automatically mining abbreviation expansions in programs to enhance software maintenance tools," *In Proceedings of the International Working Conference on Mining Software Repositories (MSR 08)*, pp. 79–88, 2008.

[20] M. Allamanis and C. Sutton, "Mining source code repositories at massive scale using language modeling," *In Proceedings of the 10th Working Conference on Mining Software Repositories (MSR 13)*, pp. 207–216, 2013.

[21] L. Yao, X. Mao, Z. Li, Z. Yang, and Y. Gang, "Internal quality assurance for external contributions in github: An empirical investigation," *Journal of Software: Evolution and Process*, vol. 30, no. 4, p. e1918, 2017.

[22] J. Jiang, D. Lo, J. He, X. Xia, P. S. Kochhar, and L. Zhang, "Why and how developers fork what from whom in github," *Empirical Software Engineering*, vol. 22, no. 1, pp. 547–578, Feb. 2017.

[23] N. Yoshida, T. Hattori, and K. Inoue, "Finding similar defects using synonymous identifier retrieval," *In Proceedings of the 4th International Workshop on Software Clones (IWSC 10)*, pp. 49–56, 2010.

[24] P. C. Rigby, Y. Zhu, S. M. Donadelli, and A. Mockus, "Quantifying and mitigating turnover-induced knowledge loss: Case studies of chrome and a project at avaya," *In Proceedings of 38th International Conference on Software Engineering (ICSE 16)*, pp. 1006–1016, 2016.

[25] J. Cohen, "A coefficient of agreement for nominal scales," *Educational and Psychological Measurement*, vol. 20, no. 1, pp. 37–46, 1960.

[26] E. Murphy-Hill, C. Parnin, and A. P. Black, "How we refactor, and how we know it," *IEEE Transactions on Software Engineering*, vol. 38, no. 1, pp. 5–18, 2012.

[27] E. Murphy-Hill and A. P. Black, "Refactoring tools: Fitness for purpose," *IEEE Software*, vol. 25, no. 5, pp. 38–44, 2008.

[28] K. Herzig and A. Zeller, "The impact of tangled code changes," *In Proceedings of 10th Working Conference on Mining Software Repositories (MSR 13)*, pp. 121–130, 2013.

[29] G. Li, H. Liu, and A. S. Nyamawe, "A survey on renamings of software entities," *ACM Computing Surveys*, vol. 53, pp. 1–38, 2020.

[30] Z. Sharafi, Z. Soh, Y.-G. Gueheneuc, and G. Antoniol, "Women and men - different but equal: On the impact of identifier style on source code reading," *In Proceedings of the International Conference on Program Comprehension (ICPC 12)*, pp. 27–36, 2012.

[31] D. Binkley, D. Lawrie, S. Maex, and C. Morrell, "Identifier length and limited programmer memory," *Science of Computer Programming*, vol. 74, no. 7, pp. 430–445, 2009.

[32] S. Butler, M. Wermelinger, Y. Yu, and H. Sharp, "Relating identifier naming flaws and code quality: An empirical study," *In Proceedings of the 16th Working Conference on Reverse Engineering (WCRE 09)*, pp. 31–35, 2009.

[33] D. Lawrie, C. Morrell, H. Feild, and D. Binkley, "What's in a name? a study of identifiers," *In Proceedings of the 14th Internal Conference on Program Comprehension (ICPC 06)*, pp. 3–12, 2006.

[34] G. Scanniello, M. Risi, P. Tramontana, and S. Romano, "Fixing faults in c and java source code: Abbreviated vs. full-word identifier names," *ACM Transactions on Software Engineering and Methodology*, vol. 26, no. 2, pp. 6:1–6:43, Jul. 2017.

[35] F. Deissenboeck and M. Pizka, "Concise and consistent naming," *Software Quality Journal*, vol. 14, no. 3, pp. 261–282, Sep. 2006.

[36] B. Lin, S. Scalabrino, A. Mocci, R. Oliveto, and M. Lanza, "Investigating the use of code analysis and nlp to promote a consistent usage of identifiers," *In Proceedings of the IEEE International Working Conference on Source Code Analysis and Manipulation*, pp. 81–90, 2017.

[37] S. Scalabrino, G. Bavota, C. Vendome, M. Linares-Vasquez, D. Poshyvanyk, and R. Oliveto, "Automatically assessing code understandability: How far are we?" *In Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*, pp. 417–427, 2017.

[38] M. Allamanis, E. T. Barr, C. Bird, and C. Sutton, "Learning natural coding conventions," *In Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 14)*, pp. 281–293, 2014.

[39] S. Butler, "Mining java class identifier naming conventions," *In Proceedings of the 34th International Conference on Software Engineering (ICSE 12)*, pp. 1641–1643, 2012.

[40] J. Falleri, M. Huchard, M. Lafourcade, C. Nebut, V. Prince, and M. Dao, "Automatic extraction of a wordnet-like identifier network from software," *In Proceedings of the IEEE International Conference on Program Comprehension (ICPC 10)*, pp. 4–13, 2010.