Published version: Naoki Doi, Yuki Osumi, Shinpei Hayashi, "RENAS: Prioritizing Co-Renaming Opportunities of Identifiers,", in Proceedings of the 40th IEEE International Conference on Software Maintenance and Evolution, pp. 562–573, 2024, https://doi.org/10.1109/ICSME58944.2024.00057

RENAS: Prioritizing Co-Renaming Opportunities of Identifiers

Naoki Doi
School of Computing
Tokyo Institute of Technology
Tokyo, Japan
doi@se.c.titech.ac.jp

Yuki Osumi
School of Computing
Tokyo Institute of Technology
Tokyo, Japan
osumi@se.c.titech.ac.jp

Shinpei Hayashi
School of Computing
Tokyo Institute of Technology
Tokyo, Japan
hayashi@c.titech.ac.jp

***Abstract*—Renaming identifiers in source code is a common refactoring task in software development. When renaming an identifier, other identifiers containing words with the same naming intention related to the renaming should be renamed simultaneously. However, identifying these related identifiers can be challenging. This study introduces a technique called RENAS, which identifies and recommends related identifiers that should be renamed simultaneously in Java applications. RENAS determines priority scores for renaming candidates based on the relationships and similarities among identifiers. Since identifiers that have a relationship and/or have similar vocabulary in the source code are often renamed together, their priority scores are determined based on these factors. Identifiers with higher priority are recommended to be renamed together. Through an evaluation involving real renaming instances extracted from change histories and validated manually, RENAS demonstrated an improvement in the F1-measure by more than 0.11 compared with existing renaming recommendation approaches.**

I. Introduction

Renaming identifiers is a crucial activity in software development, as identifiers constitute approximately 70% of the source code [1], playing a pivotal role in developers' understanding of programs [2]. Inconsistent or inappropriate identifiers can significantly hinder developers' understanding, leading them to frequently modify these identifiers to more suitable names throughout the software development process [1].

However, renaming is not an easy task [3]. When a developer decides to rename an identifier, other associated identifiers that share the same concept are often necessary to be renamed as well. This process can be time-consuming, as these identifiers may be spread across multiple files, and variations in word forms, such as inflections, must also be considered. Failing to consistently rename an identifier can result in compromised naming consistency and a decrease in the internal quality of the software.

To reduce the effort, methods that identify and recommend groups of identifiers to be renamed together have been investigated [4], [5]. For example, Liu et al. [4] specified identifiers related to the one being renamed and considered them as potential candidates for renaming, followed by the recommendation of identifiers to be renamed. This process continues with identifiers associated with those recommended for renaming, with recommendations being made until the process fails.

Merely listing identifiers that can undergo the same renaming operation is insufficient for capturing all the identifiers that necessitate renaming (see Section II for details). Therefore, it is essential to include identifiers related to identifiers to which the renaming operation cannot be applied as candidates and to exclude identifiers that should not be renamed from the list of candidates.

This study proposes the renaming recommendation approach, referred to as RENAS, which determines priority scores for identifiers based on their relationships to the renamed identifiers. By including identifiers that can be traced back through relationships in the recommendation scope, the approach identifies necessary renamings while assigning lower priority to identifiers named with different intentions. We evaluate RENAS using real-world examples of renaming. We show that RENAS is more accurate than existing approaches and that both the relationship and similarity should be taken into account for priority scores.

The main contributions of this study are as follows:

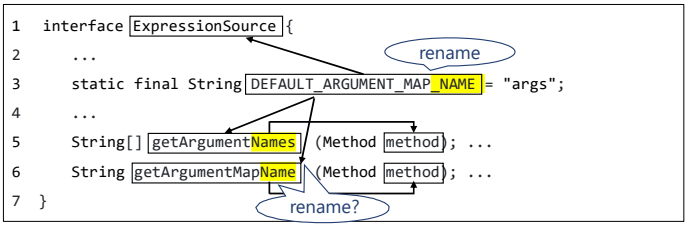
- An approach for recommending identifiers with degrees of priority.
- A dataset of manually validated real-world co-renaming examples.
- A comparative evaluation of renaming recommendation approaches across 13 open-source software projects, including the application to the manually validated dataset.

The replication package, which includes the experimental dataset and the RENAS tool, is publicly available [6], [7].

The remainder of this paper is organized as follows. Section II identifies problems of existing approaches using actual renamings. Section III explains the generation of renaming recommendations. Section IV evaluates RENAS. Section V reviews previous studies related to identifiers and their renaming practices. Section VI concludes the paper and explores potential future research directions.

II. Motivation

Renamings are a common practice in software development, although they are not always straightforward [3]. When one

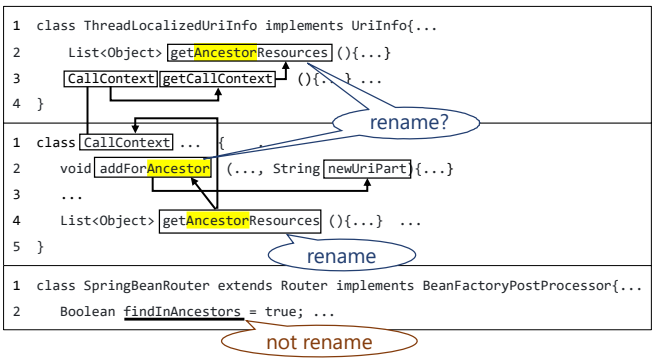


(a) Before renaming.

```
1 interface ExpressionSource {  ...
2     static final String DEFAULT_ARGUMENT_MAP_VARIABLE_NAME = "args";...
3     String[] getArgumentVariableNames(Method method); ...
4     String getArgumentMapVariableName(Method method); ...
5 }
```

(b) After renaming.

Fig. 1. Renaming in *spring-integration*.



(a) Before renaming.

```
1 class ThreadLocalizedUriInfo implements UriInfo{...
2     List<Object> getMatchedResources(){...}
3     CallContext getCallContext(){...} ...
4 }
1 class CallContext ... {     ...
2    void addForMatched(..., String newUriPart){...}
3    ...
4    List<Object> getMatchedResources(){...}  ...
5 }
1 class SpringBeanRouter extends Router implements BeanFactoryPostProcessor{...
2    Boolean findInAncestors = true; ...
```

(b) After renaming.

Fig. 2. Renaming in *restlet-framework-java*.

identifier is renamed, other identifiers named with the same intention should also be co-renamed. An example of renaming in *spring-integration*[1] is shown in Fig. 1. The source code before and after renaming are shown in Figs. 1a and 1b, respectively. The changes made before and after renaming are highlighted, showcasing the insertion of the term `variable` before the term `name`. The field name `DEFAULT_ARGUMENT_MAP_NAME`, method name `getArgumentNames`, and `getArgumentMapName` contain the word `name`, and because they indicate the same concept, the developer insert `variable` for them as well. This practice of co-renaming is quite common, as reported by Saika et al. [8], who found that renaming was the most frequent refactoring conducted simultaneously. Finding identifiers that should be co-renamed can be a time-consuming task, and overlooking such changes may lead to inconsistencies in the code.

To reduce the effort, existing approaches have been developed to recommend identifiers that should be co-renamed based on their relationships with the renamed identifier. For example, Liu et al. [4] defined relationships between identifiers and extracted candidate identifiers to which the same renaming operation can be applied for renaming based on the defined relationships with the renamed identifier, prioritizing identifiers with high similarity to the renamed identifier.

We present an example of an application of a straightforward relationship-based approach. The initial step for the source code shown in Fig. 1a involves normalizing `names` to `name` by eliminating the word form from the renaming. For example, when a developer renames `DEFAULT_ARGUMENT_MAP_NAME` to `DEFAULT_ARGUMENT_MAP_VARIABLE_NAME`, the renaming operation "insert `variable` before `name`" is obtained. From the relationships indicated as arrows in Fig. 1a, after eliminating the word form, the following candidate identifiers are obtained: `ExpressionSource`, `getArgumentName`, and `getArgumentMapName`. Subsequently, we check whether the renaming operation can be applied to these identifiers. The identifiers `getArgumentName` and `getArgumentMapName` are found to be applicable, and unexplored identifiers associated with them are added to the list of candidates. The newly identified candidates are two `method` parameters in Lines 5 and 6. However, because the renaming operation cannot be applied to them, the recommendation process is halted, resulting in the final outcome of (`getArgumentNames`, `getArgumentMapName`).

Existing approaches cannot enumerate all the identifiers that require renaming. The actual example of renaming performed in *restlet-framework-java*[2] is shown in Fig. 2. The source code before and after renaming are shown in Figs. 2a and 2b respectively, with the renamed identifiers highlighted. These renamings altered the concept of `ancestor` to `matched`.

For example, consider renaming the method `getAncestorResources` of the class `CallContext` to `getMatchedResources`. By examining the relationships indicated as arrows in Fig. 2, the potential identifiers are `CallContext` and `addForAncestor`. Additionally, because the renaming operation can be applied to `addForAncestor`, the identifier `newUriPart`, which is related to it, is also a candidate. However, the recommendation ends here because the renaming operation cannot be applied. The result consists only of `addForAncestor`, and `getAncestorResources` of class `ThreadLocalizedUriInfo` is not recommended.

To find all identifiers requiring renaming, candidates should be explored also from identifiers related to the identifiers to

[1]https://github.com/spring-projects/spring-integration/commit/5ebafd8

[2]https://github.com/restlet/restlet-framework-java/commit/2938644

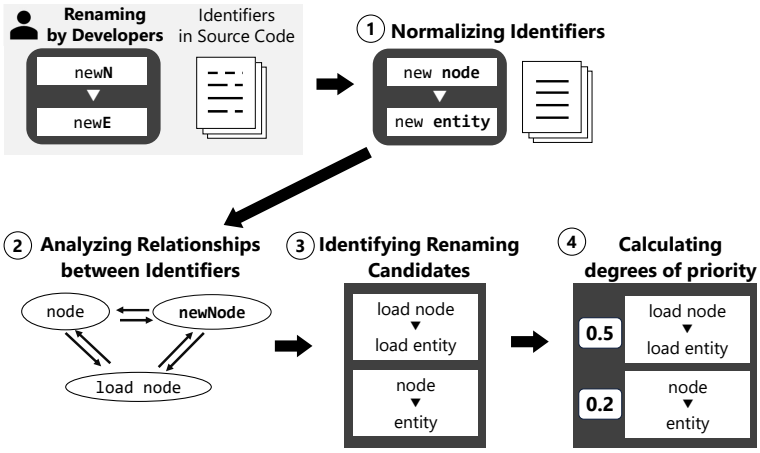


Fig. 3. Overview of RENAS.

which the renaming operation cannot be applied. By analyzing the relationships in Fig. 2a, three relationships can be traced from `getAncestorResources` of `CallContext` to `getAncestorResources` of `ThreadLocalizedUriInfo`. Consequently, by tracing relationships from the identifiers to which the renaming operation cannot be applied, candidates can include `getAncestorInfos` of `ThreadLocalizedUriInfo`.

However, excessive relationship tracing can result in inappropriate identifier recommendations. In Fig. 2a, the `findInAncestors` of the class `SpringBeanRouter` was not renamed. The `Ancestors` in `findInAncestors` was not renamed because it represents a different concept from the renamed `Ancestor` in `getAncestorResources`. The `findInAncestor` can be reached by tracing the relationship from `getAncestorResources` more than ten times. Therefore, if unlimited transitions of relationships are permitted, the recommendation of `findInAncestors` would occur.

In this study, we propose an approach called RENAS, which assigns priority to all identifiers that can be traced from the renamed identifier. This approach utilizes this priority to recommend renamings. The priority assigned is based on the similarity of identifiers and the number of traced relationships. By tracing relationships regardless of the application of the renaming operation, we can recommend identifiers that existing approaches overlook. Furthermore, by assigning priority to identifiers, those named with different intentions are given lower priority and are less likely to be recommended.

III. Methodology

A. Overview

An overview of RENAS is presented in Fig. 3. The input for this approach consists of Java source code and an identifier that has been renamed by developers. The output includes a list of identifier candidates that can be renamed and their priority levels. First, identifiers in the source code and the identifier renamed by developers are normalized, followed by the analysis of relationships between identifiers in the source code. Next, the relationships are traced from the renamed identifier. Subsequently, the renaming candidates are extracted to include identifiers to which the same renaming operation can be applied. Finally, RENAS calculates the priority of the candidates and provides recommendations. RENAS was designed to recommend renaming opportunities of identifiers for developers, rather than generating new names for identifiers. RENAS can recommend all identifiers whose priority levels are equal to or greater than a specified threshold as a set, or they can opt to receive recommendations in a ranked order based on their priority levels.

B. Normalizing Identifiers

We perform normalization by splitting identifiers, expanding abbreviations, and eliminating inflection in a specific order. This process is performed on identifiers in the source code and the renamed identifier.

We split all identifier names into lowercase word sequences. Identifier names named in *snakeCase*, *camelCase*, and *PascalCase* can be divided into a sequence of words. First, we set the delimiter characters to `$`, `_`, and numbers, and split identifier names accordingly. Next, we divide words based on *lower*, *Title*, and *UPPER* word forms. The *lower* comprises all lowercase letters, *Title* is a word with only the first letter capitalized, and *UPPER* comprises all uppercase letters. Finally, all words are then unified into lowercase.

Abbreviations are expanded using KgExpander [9] with an extension. KgExpander finds words before being abbreviated by tracing relationships between identifiers. We added a new relationship to KgExpander (details provided in the following subsection). When expanding an abbreviation, we maintain a record of the expansion, including the abbreviation, words before the abbreviation, and the source file containing the abbreviation.

When an identifier name after renaming contains an abbreviation, four steps should be followed to find words before expanding the abbreviation.

1) Searching from words in the identifier name before it was renamed.

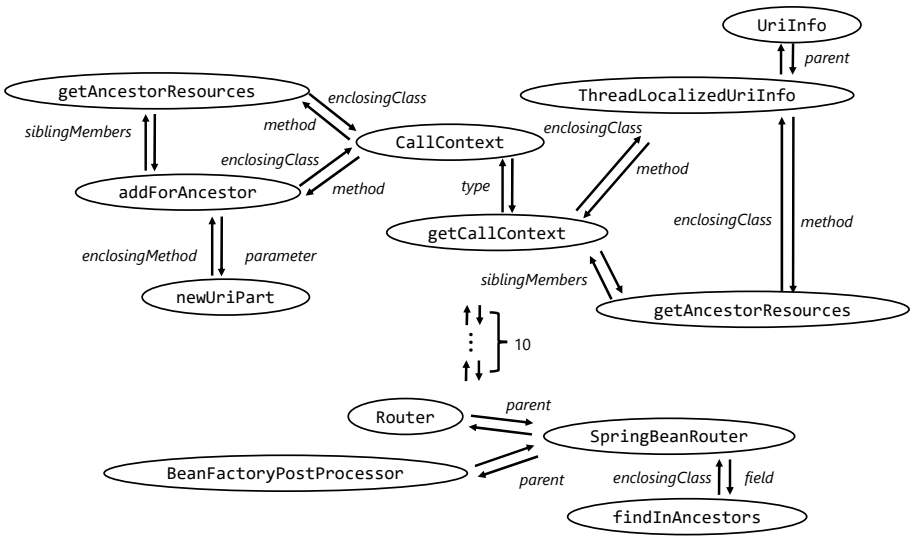


Fig. 4. Relationship graph.

2) Searching from the expansion history of the same file: Abbreviation candidates are obtained from the expansion history of the source file where renaming occurred. In the case of multiple candidates, the one with the most number of expansions is selected. Suppose that the number of expansions is the same. In that case, we select the one with the longest word length.
3) Searching by the expansion history of the entire project: We select the one with the longest word length from the entire project's history.
4) Using abbreviation dictionaries: We utilize the dictionaries provided by KgExpander. Note that we excluded commonly abbreviated words, such as PDF and UML, from these dictionaries.

Finally, NLTK[3] [10] unifies the conjugated forms of verbs with the infinitive form, plural of nouns with singular, and comparative and superlative degrees of adjectives with the positive degrees.

C. Analyzing Relationships between Identifiers

We analyze the following relationships to find identifiers to be renamed. The extraction of relationships is performed at the project level by static analysis of all the source files of a Java project before the commit, excluding those defined in external libraries, using the ASTParser of Eclipse JDT[4]. The relationships to be extracted are those used in KgExpander [9] (Inclusion, Inheritance, Assignment, and Typing) and Sibling used in RenameExpander [4].

- Inclusion
  - *field*: Relationship from a class i_c to its field i_f ($i_c \rightarrow i_f$)
  - *method*: Relationship from a class i_c to its method i_m ($i_c \rightarrow i_m$)
  - *enclosingClass*: Relationship from a field i_f and a method i_m to a class i_c enclosing it ($i_f, i_m \rightarrow i_c$)
  - *parameter*: Relationship from a method i_m to its parameter i_p ($i_m \rightarrow i_p$)
  - *enclosingMethod*: Relationship from a parameter i_p and a local variable i_v to a method i_m enclosing it ($i_p, i_v \rightarrow i_m$)
  - *pass*: Relationship from an argument (actual parameter) i_a of a method i_m to the method i_m ($i_a \rightarrow i_m$)
- Inheritance
  - *parent*: Relationship between a class i_c and its direct subclass i_{ec} or its interface i_{ic} ($i_c \leftrightarrow i_{ec}, i_{ic}$)
  - *ancestor*: Relationship between a class i_c and its ancestor class (a subclass of its subclass or more) i_{ac} ($i_c \leftrightarrow i_{ac}$)
- Assignment
  - *assignmentEquation*: Relationship between a left side value i_{left} and a right side value i_{right} of the assignment statement, such as a field, parameter, variable, or invocation of a method ($i_{left} \leftrightarrow i_{right}$)
  - *argument*: Relationship between a parameter i_p of a method i_m and an argument (actual parameter) i_a of the method i_m ($i_p \leftrightarrow i_a$)
- Typing
  - *type*: Relationship between an identifier i, such as a field, method, parameter, or variable and its type i_t ($i \leftrightarrow i_t$)
- Sibling
  - *siblingMembers*: Relationship between fields i_f of a class i_c and methods i_m of a class i_c($i_f \leftrightarrow i_f, i_m \leftrightarrow i_m, i_f \leftrightarrow i_m$)
  - *parameterOverload*: Relationship between a parameter i_{p1} of a method i_{m1} and a parameter i_{p2} of a method i_{m2} that overloads i_{m1} ($i_{p1} \leftrightarrow i_{p2}$)

[3]https://www.nltk.org/

[4]https://projects.eclipse.org/projects/eclipse.jdt

Notably, RenameExpander does not utilize Typing, and other relationships it employs are similar but not identical. For example, Inheritance only considers parent classes in RenameExpander. Additionally, we introduced a new relationship *parameterOverload* to the existing relationships used by RenameExpander. Among the methods defined in the class, the overloaded methods i_{m1} and i_{m2} exhibit functional similarities, and the identifier vocabulary used is largely identical. Consequently, we defined this relationship as a shortcut to the chain of (*enclosingMethod*, *siblingMembers*, *parameter*) to prioritize recommendations.

To illustrate these relationships, Fig. 4 displays the relationships between identifiers in Fig. 2a. The upper and lower graphs represent different segments of the relationship graph, with identifiers in the lower graph being traced back ten times from those in the upper graph. For example, the path from `addForAncestor` to `getCallContext` can be reached by following two relationships (*enclosingClass*, *type*).

D. Identifying Renaming Candidates

We extract renaming operations from the developer's renaming. When an identifier is renamed, we find other identifiers that are reachable via relationships from it and to which one of the extracted renaming operations is applicable. We consider them as potential candidates for renaming.

We define the following five types of renaming operations.

- *insert*(inserted word sequence): Insert word sequence.
  `reference`→`wordReference`: *insert*([`word`])
- *delete*(deleted word sequence): Delete word sequence.
  `wordReference`→`reference`: *delete*([`word`])
- *replace*(word sequence before replacement, word sequence after replacement): Replace word sequence.
  `termRef` → `wordRef`: *replace*([`term`], [`word`])
- *order*(word sequence before reordering, word sequence after reordering): Reorder word sequence.
  `refWord`→`wordRef`: *order*([`ref`, `word`], [`word`, `ref`])
- *format*(word before format change, word after format change): Change word format
  - $format_p$: Change the plural or singular form of a word. `word` → `words`: $format_p$(`word`, `words`)
  - $format_a$: Abbreviate word.
    `reference` → `ref`: $format_a$(`reference`, `ref`)
  - $format_c$: Change word conjugation.
    `refer` → `refering`: $format_c$(`refer`, `refering`)

When multiple renaming operations are extracted from a renaming, the renaming is considered as a candidate if at least one extracted operation is applicable. However, certain renaming operations, such as customary changes, typo corrections, changes in the structure of identifier names, and expansion of abbreviations do not adhere to the relationships [5]. Therefore, we exclude renaming operations such as *order*, $format_a$, or $format_c$ from the recommendations. The applicable conditions for each operation are as follows:

- *insert*: The inserted word sequence matches either before or after.
- *delete*: Deleted word sequence exists.
- *replace*: Word sequence before replacement exists.
- *order*: Two or more words are contained in a word sequence before reordering, and the order of appearance differs.
- *format*: Word before format change exists.

We define the set of renaming refactorings as $R_{\text{name}} = \{r_0, r_1, \ldots, r_{n-1}\}$, and $Op(r)$ represents the renaming operations in the refactoring r. Let $S_{op} = \{r \in R_{\text{name}} \mid op \in Op(r)\}$ be the co-renamed set when the renaming operation is common. If two or more renaming operations exist, the renaming belongs to more than one set.

E. Calculating Priority of Renaming Candidates

We determine the priority of identifiers within the renaming candidates. The higher the priority, the more the identifier should be renamed.

We calculate the priority based on the similarity and relationship of identifiers. In Section III-E1, we investigate the similarity between identifiers to highlight its significance in determining the priority for recommending co-renaming of identifiers. In addition, we calculate the priority considering relationships because identifiers connected by relationships are often co-renamed [11].

1) Calculating Priority Based on Similarity: The similarity between the renamed identifier I_{change} and candidate identifier I_{cand} (both normalized) is computed using the Dice coefficient, which is based on the similarity of constituent words. This coefficient is preferred over other similarity indices to highlight shared vocabulary. The similarity score is defined as follows:

$$\begin{aligned} Score_{\text{sim}}(I_{\text{change}}, I_{\text{cand}}) &= Dice(I_{\text{change}}, I_{\text{cand}}) \\ &= \frac{2 * |I_{\text{change}} \cap I_{\text{cand}}|}{|I_{\text{change}}| + |I_{\text{cand}}|}. \end{aligned} \tag{1}$$

The co-renamed identifiers exhibited high $Score_{\text{sim}}$. Using RefactoringMiner 2.0.2 [12], we generated co-renamed sets S_{op} from the renaming data. The names of four projects used in this investigation, the number of co-renamed sets, and the number of renamed identifiers are shown in Table I. To ensure a diverse range of projects for the primary evaluation, four projects were randomly selected from the dataset of Osumi et al. [11]; coincidentally, two projects were small, whereas the remaining two were voluminous. Although the number of projects may not be extensive, we believe they are highly effective in promoting the utilization of $Score_{\text{sim}}$ and the threshold settings surveyed in Section III-E3.

We calculated $Score_{\text{sim}}$ of identifier names prior to renaming for all combinations $\{(r_i, r_j) \mid r_i, r_j \in S_{op} \wedge i < j\}$ within the set. The result is shown in the histogram in Fig. 5. The vertical axis represents the relative frequency, whereas the horizontal axis represents $Score_{\text{sim}}$ across 11 classes $\{[x/10, (x+1)/10) \mid 0 \le x \le 10, x \text{ is an integer}\}$. This figure illustrates a positive correlation between higher $Score_{\text{sim}}$ values and increased relative frequency, suggesting that co-renamed identifiers often exhibit high $Score_{\text{sim}}$.

TABLE I
FOUR PROJECTS UTILIZED IN THE INVESTIGATION

Project	# renames	# co-renamed sets
cordova-plugin-local-notifications	40	19
morphia	1,309	260
spring-integration	3,652	995
baasbox	36	13

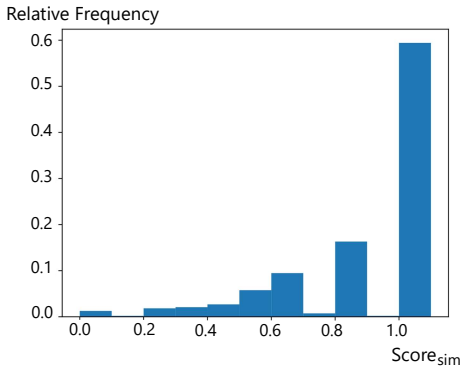


Fig. 5. Distribution of $Score_{\text{sim}}$.

2) Calculating Priority Based on Relationships of Identifiers: To calculate the distance on the relationship graph between the renamed identifier and renaming candidate, we sum the cost weights within the relationship graph as follows:

$$distance = \sum_{rel \in Rel} Cost(rel) \tag{2}$$

where $Rel = \{rel_1, rel_2, \dots\}$ represents the set of relationships traced on the shortest path, and *Cost* represents the cost function based on the type of relationship, as shown in Table II. We established four distinct *Cost* function based on the frequency of co-renaming [11]. Furthermore, we categorized the relationships of *pass*, *ancestor*, and *parameterOverload*, which were not included in the result by Osumi et al. [11], by employing the same categories as other similar relationships.

The priority based on the relationship is obtained as the inverse of *distance*, prioritizing the shortest distance:

$$Score_{\text{rel}} = \frac{1}{\text{distance}}. \tag{3}$$

3) Calculating Overall Priority: The overall priority of an identifier, denoted as *Score*, is determined by a linear combination of two priorities. The priority can be calculated as follows:

$$Score = \alpha\, Score_{\text{sim}} + (1 - \alpha)\, Score_{\text{rel}}. \tag{4}$$

TABLE II
PRIORITY OF EACH RELATIONSHIP

Cost	Relationship
1	*assignmentEquation*, *siblingMembers*, *parameterOverload*
2	*argument*
3	*parameter*, *enclosingMethod*, *pass*, *parent*, *ancestor*, *type*
4	*field*, *method*, *enclosingClass*

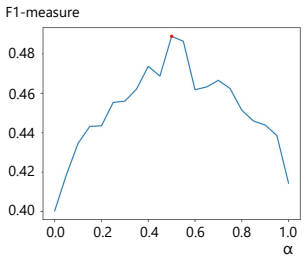


Fig. 6. Maximum F1-measure for each α in eq. (4).

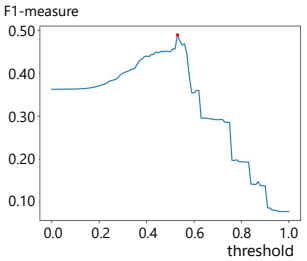


Fig. 7. F1-measure at each threshold β where $\alpha = 0.5$.

The parameter α represents the mixing ratio of $Score_{\text{rel}}$ and $Score_{\text{sim}}$. The recommendation result can manifest as either a ranking based on priority or a set of identifiers with a priority equal to or greater than the threshold β. The parameters α and β were defined empirically. Based on the approach in Section III-E1, we generated co-renamed sets for the four projects in Table I and examined whether one renaming in the set could recommend other renamings. Based on $\alpha \in \{0, 0.05, 0.10, \dots, 1\}$ and $\beta \in \{0, 0.05, 0.10, \dots, 1\}$, the results obtained by executing RENAS are shown in Figs. 6 and 7. The largest F1-measure among all β trials at a particular α is shown in Fig. 6. From this graph, an optimal value of $\alpha = 0.5$ was selected, corresponding to an F1-measure of 0.489. The F1-measure at each β when $\alpha = 0.5$ is shown in Fig. 7. Based on the results, $\beta = 0.53$ was adopted, taking the maximum value.

4) Example: We illustrate an example calculation when the method `getAncestorResources` in the class `CallContext` in Fig. 2a is renamed to `getMatchedResources`. Upon analyzing the relationship graph shown in Fig. 4, three identifiers exist that could apply the renaming operation:

- `addForAncestor`: (*siblingMembers*),
- `getAncestorResources`: (*enclosingClass*, *type*, *siblingMembers*),
- `findInAncestors`: (*enclosingClass*, *type*, ...).

The priority of each identifier is then calculated. For example, when determining the priority of `addForAncestor`, the *Dice* between [`add`, `for`, `ancestor`] and [`get`, `ancestor`, `resource`] is $1 \times 2/(3+3) = 0.333$. Therefore, the priority $Score_{\text{sim}}$ calculated from the *Dice* is 0.333. The relationship *siblingMembers* is traced from `ancestorResources` to `addForAncestor`. Because the *Cost* of *siblingMembers* is 1, the *distance* calculated from the relationship is 1, resulting in a priority $Score_{\text{rel}}$ of 1, as per eq. (3). Finally, the overall priority *Score* was determined to be $0.5 \times 0.333 + 0.5 \times 1 = 0.667$ as per eq. (4). Similarly, because $Score_{\text{sim}}$ of `getAncestorResources` is $2 \times 3/(3+3) = 1$, the *distance* of the relationship is $4 + 3 + 1 = 8$ and $Score_{\text{rel}}$ is $1/8 = 0.125$. Therefore, *Score* is $0.5 \times 1 + 0.5 \times 0.125 = 0.563$. Furthermore, $Score_{\text{sim}}$ of `findInAncestors` is $2 \times 1/(3+3) = 0.333$ and the *distance* of the relationship is $4+3+\cdots \geq 8$ and $Score_{\text{rel}} \leq 1/8 = 0.125$. Therefore, $Score \leq 0.5 \times 0.333 + 0.5 \times 0.125 = 0.229$.

In the case of renaming recommendation using the thresh-

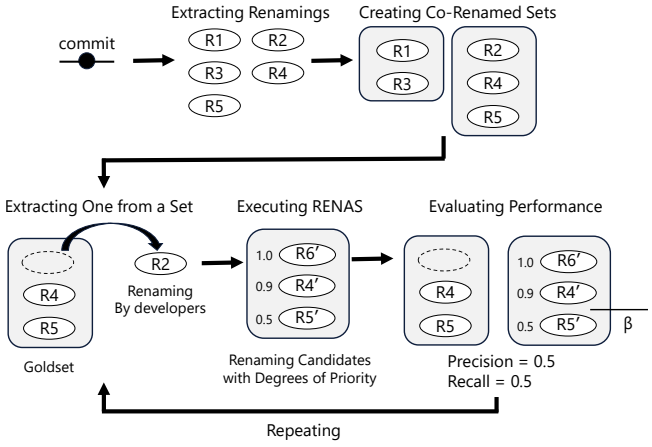


Fig. 8. Overview of the evaluation process.

old β, the priority of `addForAncestor` and `getAncestorResources` exceeds the threshold of 0.53, resulting in two renaming recommendations being generated.

IV. Evaluation

In this section, we evaluate RENAS by answering the two research questions.

- RQ1: What is the performance of RENAS compared with that of existing approaches?
- RQ2: How does the performance vary depending on how to prioritize?

RQ1 aims to demonstrate that RENAS outperforms existing methods in terms of renaming recommendations. In answering RQ1, we utilize the set of recommendations obtained using the threshold β. RQ2 focuses on the validity of calculating priority based on identifier similarity and relationships. RQ2 utilizes recommendation ranking based on priority.

An overview of the evaluation process is shown in Fig. 8. We specifically considered renames within a single commit, so only one developer is involved in the renames. The dataset of co-renamed sets in Section IV-A was utilized to investigate the potential for one renaming in a set to recommend other renamings.

A. Data Collection

Two datasets were prepared from GitHub projects as of December 2023: one was automatically identified and the other was manually validated.

The automatically identified dataset was created by selecting 11 projects from the renaming study conducted by Osumi et al. [11]. Co-renamed sets were generated using RefactoringMiner 2.0.2 [12], defined in Section III-D when four or more renamings were obtained in a single commit. This lower limit was set to emphasize the situations of overlooking co-renamings. These projects yielded 14,607 renamings and 3,361 co-renamed sets.

The manually validated dataset was developed by selecting two projects from Osumi et al., as outlined in Table III. This dataset comprised sets of co-renamed items that were manually validated based on the rename refactorings identified using RefactoringMiner 2.0.2. Manual validation was conducted by an author.

TABLE III
Projects to Validate Manually

Name	# Java files	LOC	# renamings	# co-renamed sets
ratpack	831	42,160	1,423	327
ArgoUML	1,884	174,574	709	178

To create the manually validated dataset, the first step involved checking whether the renamings obtained from the project adhered to the specified relationship. As described in Section III-D, customary changes, typo corrections, changes in the structure of identifiers, and expansion of abbreviations are excluded from the dataset because they do not follow the relationships. Only renamings that followed the relationships were included in the co-renamed set.

In cases where an abbreviation was present in the renaming, efforts were made to identify the original words that were abbreviated within the program. Suppose that the original words could not be located within the program. In that case, we identified original words by referring to studies on identifier abbreviation [13], [14]. Following the expansion of the abbreviation, we determined whether the renaming followed the relationship.

The creation of a manually validated dataset enabled the exclusion of renamings that did not conform to the relationship, which the automatically identified dataset could not exclude. Additionally, the manual validation process was necessary to address the possibility of inaccuracies in the extracted renaming operations and errors in the formation of co-renamed sets. There are three examples of renamings that were manually excluded: `FigSeperator` → `FigSeparator`, `GoElementToDependentElement` → `GoElement2DependentElement`, and `token` → `theToken`. The first renaming operation was extracted as *replace*([`seperator`], [`separator`]) by RENAS. However, it was determined to be a typo correction and therefore excluded from the dataset. RENAS extracted the second renaming operation as *delete*([`to`]). However, this operation was also excluded as it only altered the structure of the identifier. The third renaming operation was *insert*([`the`]). This operation was excluded from the dataset as it was deemed a customary change based on observations of the renaming of inserting "the" before a noun in many commits in the project.

This dataset was utilized to assess the precision of automatically identified results.

B. RQ1: What is the performance of RENAS compared with that of existing approaches?

1) Study Design: The existing approach recommends renamings by tracing the relationships between identifiers until the renaming operation is no longer applicable. In contrast, RENAS recommends all identifiers for which the relationships can be traced. We investigated the performance differences resulting from varying recommendation approaches, utilizing precision, recall, and F1-measure as evaluation metrics.

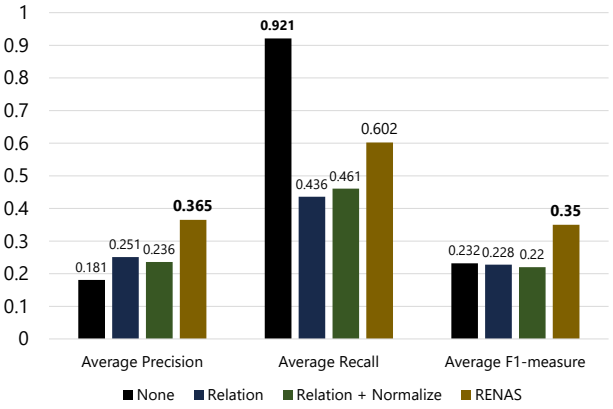


Fig. 9. RQ1: Results of the automatically identified dataset.

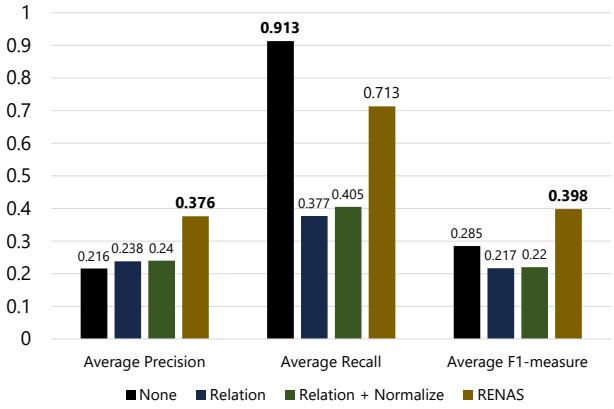


Fig. 10. RQ1: Results of the manually validated dataset.

RENAS was compared with *None*, *Relation*, and *Relation+Normalize*.

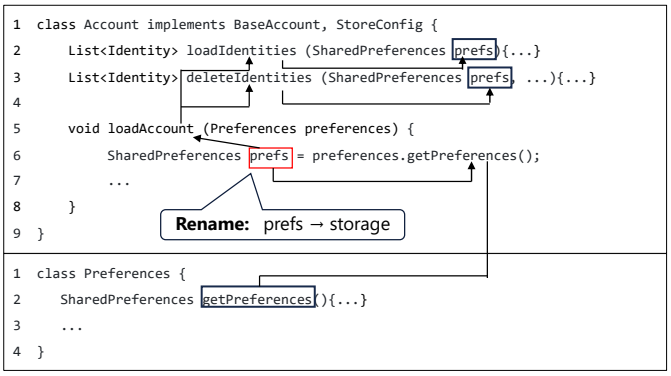
- *None*: Recommending from all identifiers in the source code without considering word inflections and abbreviations.
- *Relation*: Adding identifiers related to the renamed identifier to the candidates and recommending identifiers to which the renaming operation can be applied from the candidates. This method also includes identifiers related to those eligible for renaming. Word inflections and abbreviations were not considered. This setting was designed to mimic RenameExpander [4] for comparison with the proposed approach.[5]
- *Relation+Normalize*: This considers word inflections and abbreviations in the *Relation* recommendation approach.
- *RENAS* (Proposed approach): This approach recommends identifiers with a priority equal to or exceeding the specified threshold.

2) Results and Discussion: The evaluation results based on the automatically identified dataset are shown in Fig. 9. Each value in this figure represents the average across all projects.

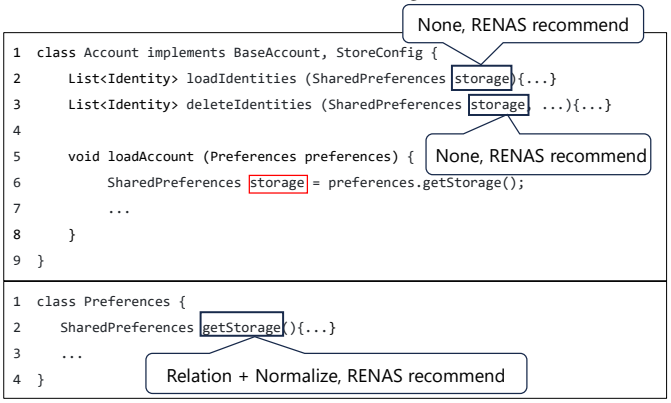
The *None* approach achieved the highest recall among the four approaches as they recommend renaming from all identifiers in the source code. *RENAS* achieved the highest precision, and its recall was higher than that of *Relation* and *Relation+Normalize*. This can be attributed to the fact that *RENAS* considered identifier similarity and suggested renamings for all identifiers that can be traced back to relationships. For example, identifiers that could not be recommended by tracing the identifiers to which the renaming operation could be applied are shown in Fig. 2. The performance of *RENAS* was higher owing to the abundance of such renamings. Overall, the F1-measure of *RENAS* surpassed those of other approaches.

The results for the manually validated dataset are shown in Fig. 10. *RENAS* demonstrated the highest precision and F1-measure, whereas *None* exhibited the highest recall. Therefore, we conclude that *RENAS* outperformed the existing approaches.

[5]RENAS cannot be directly compared with RenameExpander owing to their differences in the implementation infrastructure, relationships utilized, and the basic design of the recommendation approach. To facilitate a comparison focusing on the differences in the scoring approach by RENAS while unifying other minor settings, this evaluation setting has been devised.



(a) Before renaming.



(b) After renaming.

Fig. 11. Recommendation example in *thunderbird-android*.

In our analysis of successful rename recommendations, we distinguished between identifiers that were *directly* traceable (recommended by *Relation+Normalize*) and those that were *indirectly* traceable (recommended by *RENAS* with $\beta = 0$ but not by *Relation+Normalize*). Consequently, we observed 87,349 directly and 85,007 indirectly traceable identifiers. These findings underscore the importance of considering indirectly connected identifiers when recommending renamings.

An example of successful recommendations for each approach is shown in Fig. 11. The source code of *thunderbird-android*[6] before and after renaming are shown in Figs. 11a and 11b, respectively. Consider the case in which a developer renames `prefs` to `storage` in Line 6. In this commit, the parameter `prefs` in Lines 2 and 3 and the method `getPreferences` in Line 2 were renamed. In the case of *None*, the parameter `prefs` in Lines 2 and 3 could be recommended, whereas `getPreferences` could not be recommended because identifiers were not normalized. Conversely, in the case of *Relation*, the extracted renaming operation was *replace*([`prefs`], [`storage`]) because normalization was not performed. The identified candidates for renaming were `loadAccount` and `getPreferences`. However, no recommendation was made as neither of them contained `prefs`. `SharedPreferences` was defined in an external library, which was outside of the scope. In the case of *Relation+Normalize*, the extracted renaming operation was *replace*([`preference`], [`storage`]) owing to normalization. This led to the recommendation of renaming `getPreferences` to `getStorage`. In the case of *RENAS*, the extracted renaming operation was *replace*([`preference`], [`storage`]). The parameter `prefs` on Lines 2 and 3, as well as the method `getPreferences` on Line 2 could be traced through relationships and were recommended because their computed priority exceeded the threshold.

In some cases, *Relation+Normalize* succeeded in recommending a renaming, whereas *RENAS* failed. For example, when multiple renaming occurred in a chain, *Relation+Normalize* could recommend all the renamings, whereas *RENAS* failed owing to low priority when the relationship was traced more than once. The RENAS method may benefit from adjusting the eq. (3) of the relationship to enhance its performance.

> *RENAS* achieved the highest F1-measure and precision of 0.35 and 0.365, respectively. *None* reached the highest recall at 0.921. We obtained similar results for the manually validated dataset. For further performance enhancement, a method for calculating the priority of relationships should be developed.

C. RQ2: How does the performance vary depending on how to prioritize?

1) Study Design: We sorted the renaming candidates obtained by running RENAS in order of priority and measured the mean average precision (MAP), mean reciprocal rank (MRR), and Top 1, 5, 10-recall for $\alpha \in \{0, 0.05, 0.10, \ldots, 1\}$ of eq. (4). When $\alpha = 0$, prioritization is based solely on $Score_{\mathrm{rel}}$, whereas when $\alpha = 1$, prioritization is determined solely by $Score_{\mathrm{sim}}$. This evaluation aimed to determine the impact of considering both relationships and similarities in prioritization.

In cases where two priorities were equal, we utilized a unique ID as a secondary sorting criterion to establish a complete ranking order for evaluating the effects of prioritization. The ID is a string comprising the Java file name, identifier name, and type.

[6]https://github.com/thunderbird/thunderbird-android

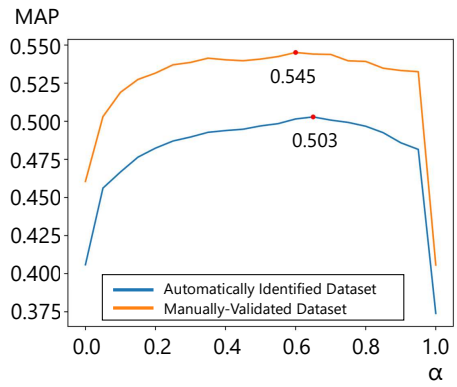


Fig. 12. MAP of the automatically identified and manually validated dataset.

2) Results and Discussion: The graph of the MAP is shown in Fig. 12, representing the results from the automatically identified dataset and manually validated dataset. The horizontal axis of Fig. 12 represents α, whereas the vertical axis represents the MAP. From the results of the automatically identified dataset, as α increased from 0, the value of MAP also increased. When α was 0.65, the highest value of MAP was 0.503. The value of MAP decreased as α increased from 0.65, and at $\alpha = 1$, the value of MAP rapidly decreased. Similar results were obtained for the manually validated dataset.

The values of the other evaluation metrics in the automatically identified and manually validated datasets are shown in Figs. 13 and 14, respectively. Although the highest value of α differed, the overall trends in the evaluation metrics remained consistent.

The study revealed that prioritizing recommendations based on both similarity and relationships proved to be more effective across all projects compared with recommendations based solely on similarity or relationships.

RENAS may fail to make successful recommendations for two reasons.

- *The recommended identifier has been deleted after the commit.* If developers deleted the identifier recommended for renaming after the commit, the recommendation was deemed unsuccessful. For example, if the developer renamed `ancestorResources` → `matchedResources`, we obtained *replace*([`ancestor`], [`matched`]). We recommended `ancestorURIs` because the identifier existed when our approach traced the relationship. However, this renaming recommendation failed because the modification at the commit also deleted `ancestorURIs`.
- *Missing relationship.* Despite our efforts to identify renaming candidates by tracing relationships from the renamed identifier, instances existed in which our approach failed to include certain identifiers in the list of candidates, no matter how extensively we traced the relationships. To recommend them, defining new types

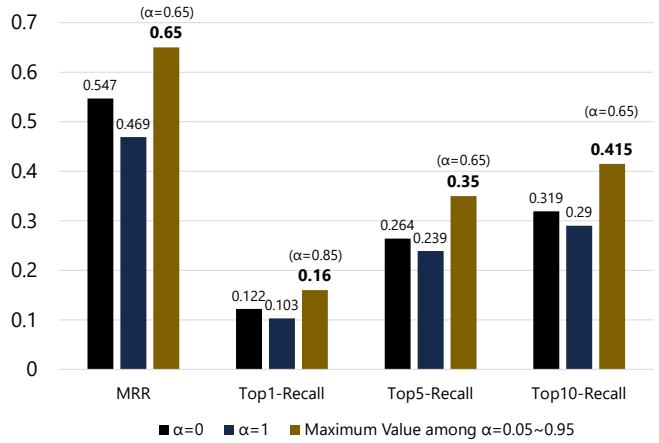


Fig. 13. RQ2: Result of the automatically identified dataset.

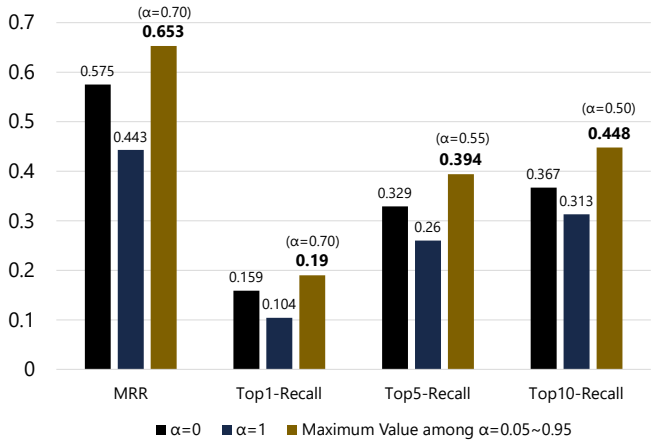


Fig. 14. RQ2: Result of the manually validated dataset.

of relationships is imperative. An example of such a missing relationship is the co-renaming of "classes in the same package". Introducing such a relationship could potentially enhance recall while reducing precision.

For each evaluation metric, $\alpha \in \{0, 1\}$ produced the lowest value, and the highest value was produced within the range of $\alpha \in [0.5, 0.85]$. Similar results were obtained for the manually validated dataset. The similarity and relationship between identifiers should be considered when calculating the priority.

D. Threats to Validity

1) Internal Validity: The performance of RENAS may have been influenced by the accuracy of component tools, such as NLTK, KgExpander, and RefactoringMiner. For example, RefactoringMiner may generate incorrect renames or lack correct renames, resulting in a decrease in precision. KgExpander and NLTK may not correctly normalize identifiers. For example, if a developer renames `node` to `entity`, a variable `n` (as an abbreviation of "node") may be the target to be co-renamed. KgExpander might mistakenly interpret an identifier of `n` as an abbreviation for "number" instead of "node", which results in a lack of recommendation.

The automatically identified dataset may not be accurate if the detected renaming operation is incorrect. To address this concern, we curated a manually validated dataset. However, a bias may be included in the dataset because the validation was conducted by a single person, potentially resulting in irrelevant renames being included. As the dataset is publicly available [6], [7], researchers are encouraged to investigate the impact of this potential threat.

Developers may not have made all necessary renamings in the commits utilized for the evaluation. Failure to make these renamings could lead to misclassification of true positives to false positives in evaluating renaming recommendations.

2) External Validity: We evaluated RENAS on 13 projects, including manual inspection, and demonstrated that RENAS outperformed existing approaches. Because the number of projects may be insufficient, we encourage further evaluation on a larger scale to showcase the effectiveness of our approach.

The demonstration of RENAS was conducted only for the Java programming language, and its applicability to other programming languages should be evaluated in future studies. We believe that RENAS could be applied to other Java-like object-oriented programming languages because similar relationships could be extracted.

V. Related Work

Numerous studies on the impact of identifier names have been conducted, revealing that names with detailed roles are crucial in understanding programs. Feitelson et al. [15] surveyed 334 developers to understand how identifier names are chosen and developed a model for constructing identifier names. Their findings indicated that identifiers named under similar circumstances varied among developers. Adhering to the model could lead to the generation of better identifier names. Hofmeister et al. [16] investigated the influence of identifier names with single letters or abbreviations on program comprehension. They demonstrated that code comprehension was conducted 19% faster for unabbreviated words. Beniamini et al. [14] investigated the concepts strongly associated with variables named with a single letter. Avidan et al. [17] investigated the impact of variable names on program comprehension, highlighting the importance of meaningful names in aiding comprehension. They noted that parameter names had a more significant influence on program comprehension compared with local variable names. Some studies have investigated the characteristics of identifiers utilized in large datasets. Zhang et al. [18] surveyed 5,000 projects on part-of-speech tags, identifier length, and initialization. They found that nouns, verbs, and adjectives were frequently utilized in identifiers, with identifier length correlating with importance. Additionally, they observed that over 40% of field and variable names were not initialized at declaration time. Feitelson et al. [19] examined the distribution of the name length and scope of variable identifiers of Java projects.

TABLE IV
APPROACHES FOR RECOMMENDING RENAMINGS

	Relationships	Abbreviations	Inflection Words
IDD [1]			
Thies et al. [21]	✓		
Zhang et al. [5]	✓+	✓	✓
RenameExpander [4]	✓++		
RENAS	✓++	✓	✓

Their analysis of data from approximately 1,000 Java projects revealed that the wider the scope, the more words incorporated into identifiers, and words comprising six or more letters were often abbreviated.

Developers frequently rename identifiers because they are important, and inappropriate names impair program comprehension. Arnaoudova et al. [3] demonstrated that renaming identifiers is a complex task, categorizing them into various types based on factors, such as identifier type, change operation, semantic change, and grammatical change. They further defined different renaming operations based on this taxonomy. Peruma et al. [20] examined 3,795 Java projects to investigate the types of renamings that occurred in class, method, and package names. Their findings indicated that the meaning of an identifier often became more specific through renaming, and the grammatical structure of identifiers frequently changed. Moreover, they highlighted that developers' change intentions could be simple or complex.

The characteristics of the existing approaches and our approach for recommending rename refactorings are listed in Table IV. The checkmark (✓) in the table indicates that the element in the column is considered in the approach. Additionally, the plus sign (+) in the "Relationships" column indicates the number of relationship types, with more pluses indicating more relationships. Deibenbock et al. [1] developed a tool named Identifier Dictionary (IDD) to maintain consistent and concise identifier names. IDD stores information on all identifiers in the source code and recommends renaming them if it detects identifiers with the same name but different types. Thief et al. [21] recommended renaming identifiers based on the assignment relationship. They made recommendations to developers by identifying misspellings, synonyms, and incorrect naming from graphs based on the assignment relationship. Liu et al. [4] established relationships between identifiers and specified candidate identifiers to be renamed based on their relationships with the identifier renamed by a developer. Among the candidates, they recommended renaming identifiers to which the renaming operation could be applied in the order of the similarity of identifiers. Zhang et al. [5] utilized a score obtained through machine learning derived from identifier relationships, identifier information, and files containing the identifiers. They also incorporated a score calculated from previous renaming. RENAS performed recommendation by defining the relationship between identifiers in more detail compared with that of Zhang et al. However, it does not consider information about files and previous renaming. Prioritizing identifiers while considering these factors may lead to enhanced accuracy in recommendations.

Research has been conducted on prioritizing identifier name recommendations. Kasegn et al. [22] conducted a study to determine the proximity of words within a document, and when the prefix of an identifier name was provided, their approach recommended the words connected to the prefix in order of their priority. The recommended identifiers included classes, methods, and fields. Abebe et al. [23] analyzed the concepts represented by identifiers, established relationships related to concepts, such as the *isA* and *hasProperty* relationships, and constructed an identifier relationship graph. Based on this graph, they prioritized their recommendations. Because RENAS does not rely on concept-based relationships, analyzing these relationships in addition to the current relationships and assigning priority to them may enhance the accuracy.

VI. CONCLUSION

This study proposed an approach named RENAS, which aimed to recommend renaming opportunities by assigning priorities to identifiers based on their similarities and relationships. The evaluation of RENAS involved comparing its performance with three distinct settings: *None*, *Relation*, and *Relation+Normalize* using an automatically identified dataset and a manually validated dataset. The results showed that the F1-measure of RENAS was 0.35, which was 0.11 higher than that of the other approaches. We observed similar values for the manually validated dataset. This result suggests that RENAS outperformed existing approaches in terms of effectiveness. Furthermore, we obtained higher values for each evaluation metric when considering both factors compared with when considering only similarity or relationships.

Several avenues for future research can be explored.

- *Creating a Renaming Dataset.* In our evaluations, we created co-renamed datasets from commits of projects on GitHub. However, developers may not make all the necessary renamings within a commit. To evaluate renaming recommendations more accurately, creating an accurate dataset of renamings is necessary.
- *Relevance Feedback.* Implementing re-prioritization based on developers' adoption of recommended renamings can enhance the accuracy of prioritization.
- *Detailed Investigation of Renamings.* We will conduct an empirical study to investigate the types of effective co-renaming relationship that impacts the accuracy of co-renaming recommendations and define the relationships based on it. Additionally, we will investigate the frequency and reasons for renamings in relation to the difference between direct and indirect relationships.

ACKNOWLEDGMENTS

This study was partly supported by JSPS Grants-in-Aid for Scientific Research Nos. JP24H00692, JP23K24823, JP21H04877, JP21K18302, and JP21KK0179.

References

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

[2] A. Schankin, A. Berger, D. V. Holt, J. C. Hofmeister, T. Riedel, and M. Beigl, "Descriptive compound identifier names improve source code comprehension," in *Proceedings of the 26th IEEE/ACM International Conference on Program Comprehension (ICPC 2018)*, 2018, pp. 31–40.

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

[4] H. Liu, Q. Liu, Y. Liu, and Z. Wang, "Identifying renaming opportunities by expanding conducted rename refactorings," *IEEE Transactions on Software Engineering*, vol. 41, no. 9, pp. 887–900, 2015.

[5] J. Zhang, J. Luo, J. Liang, L. Gong, and Z. Huang, "An accurate identifier renaming prediction and suggestion approach," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 6, pp. 148:1–51, 2023.

[6] N. Doi, Y. Osumi, and S. Hayashi, "Online appendix of "RENAS: Prioritizing co-renaming opportunities of identifiers"," Zenodo, https://doi.org/10.5281/zenodo.13164183, 2024.

[7] ——, "RENAS toolkit," https://github.com/salab/RENAS, 2024.

[8] T. Saika, E. Choi, N. Yoshida, A. Goto, S. Haruna, and K. Inoue, "What kinds of refactorings are co-occurred? An analysis of Eclipse usage datasets," in *Proceedings of the 6th International Workshop on Empirical Software Engineering in Practice (IWESEP 2014)*, 2014, pp. 31–36.

[9] Y. Jiang, H. Liu, and L. Zhang, "Semantic relation based expansion of abbreviations," in *Proceedings of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2019)*, 2019, pp. 131–141.

[10] S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*. O'Reilly Media, Inc., 2009.

[11] Y. Osumi, N. Umekawa, H. Komata, and S. Hayashi, "Empirical study of co-renamed identifiers," in *Proceedings of the 29th Asia-Pacific Software Engineering Conference (APSEC 2022)*, 2022, pp. 71–80.

[12] N. Tsantalis, A. Ketkar, and D. Dig, "RefactoringMiner 2.0," *IEEE Transactions on Software Engineering*, vol. 48, no. 3, pp. 930–950, 2022.

[13] C. D. Newman, M. J. Decker, R. S. AlSuhaibani, A. Peruma, D. Kaushik, and E. Hill, "An open dataset of abbreviations and expansions," in *Proceedings of the 35th IEEE International Conference on Software Maintenance and Evolution (ICSME 2019)*, 2019, p. 280.

[14] G. Beniamini, S. Gingichashvili, A. K. Orbach, and D. G. Feitelson, "Meaningful identifier names: The case of single-letter variables," in *Proceedings of the 25th IEEE/ACM International Conference on Program Comprehension (ICPC 2017)*, 2017, pp. 45–54.

[15] D. G. Feitelson, A. Mizrahi, N. Noy, A. B. Shabat, O. Eliyahu, and R. Sheffer, "How developers choose names," *IEEE Transactions on Software Engineering*, vol. 48, no. 1, pp. 37–52, 2022.

[16] J. Hofmeister, J. Siegmund, and D. V. Holt, "Shorter identifier names take longer to comprehend," in *Proceedings of the 24th IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER 2017)*, 2017, pp. 217–227.

[17] E. Avidan and D. G. Feitelson, "Effects of variable names on comprehension: An empirical study," in *Proceedings of the 25th IEEE/ACM International Conference on Program Comprehension (ICPC 2017)*, 2017, pp. 55–65.

[18] J. Zhang, S. Liu, J. Luo, J. Liang, and Z. Huang, "Exploring the characteristics of identifiers: A large-scale empirical study on 5,000 open source projects," *IEEE Access*, vol. 8, pp. 140 607–140 620, 2020.

[19] D. G. Feitelson, "Reanalysis of empirical data on Java local variables with narrow and broad scope," in *Proceedings of the 31st IEEE/ACM International Conference on Program Comprehension (ICPC 2023)*, 2023, pp. 227–236.

[20] A. Peruma, M. W. Mkaouer, M. J. Decker, and C. D. Newman, "An empirical investigation of how and why developers rename identifiers," in *Proceedings of the 2nd International Workshop on Refactoring (IWoR 2018)*, 2018, pp. 26–33.

[21] A. Thies and C. Roth, "Recommending rename refactorings," in *Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering (RSSE 2010)*, 2010, pp. 1–5.

[22] S. A. Kasegn and S. L. Abebe, "Spatial locality based identifier name recommendation," in *Proceedings of the 3rd International Conference on Information and Communication Technology for Development for Africa (ICT4DA 2021)*, 2021, pp. 65–70.

[23] S. L. Abebe and P. Tonella, "Automated identifier completion and replacement," in *Proceedings of the 17th European Conference on Software Maintenance and Reengineering (CSMR 2013)*, 2013, pp. 263–272.