2017 24th Asia-Pacific Software Engineering Conference Workshops

An Exploratory Study on Interface Similarities in Code Clones

Md Rakib Hossain Misu
Institute of Information Technology
University of Dhaka
Dhaka, Bangladesh
Email: bsse0516@iit.du.ac.bd

Abdus Satter
Institute of Information Technology
University of Dhaka
Dhaka, Bangladesh
Email: bit0401@iit.du.ac.bd

Kazi Sakib
Institute of Information Technology
University of Dhaka
Dhaka, Bangladesh
Email: sakib@iit.du.ac.bd

***Abstract*—Code cloning is one of the most popular code reusing techniques where similar pieces of code are replicated within or between code repositories. Interface similarity is a kind of replication that refers to the similarity of method names, return types and parameter types which repeat across the code repositories. Two methods with similar interfaces are prone to be cloned if those perform analogous functions either entirely or at least partially. An exploratory study is performed in this paper, to explore the relationship and effects of interface similarity in code clones. It is investigated that interface similarity can be helpful for code clone detection. First, clone methods are detected in code repositories. Then, interface information is extracted from source code and several interface similarities are measured using that information. The experimental corpus contains three different types of code repositories with 35, 109 and 24,558 Java projects respectively. The detected clone pairs in three code repositories are, on average 57,457, 102,745 and 123,576 respectively. Promising results are found as it shows on average 87.91% intra-project and 59.17% inter-project clones contain similar interfaces (i.e., return types and at least one keyword and one parameter type are similar). Besides, the average similarity of interfaces in Type-1, Type-2 and Type-3 clones are 100%, 83.47% and 81.90% respectively. These results prove the strong relationship between code clones and interface similarities. In future, it can be useful to design and develop interface driven clone detection tools.**

Keywords-Code Clone, Clone Detection, Exploratory Study

I. Introduction

Code clones are pairs of code fragments that are identical to each other and replicated within or between the code repositories. It occurs when developers reuse code through copying and pasting, automatic code generation or plagiarism with or without modifications [1]. Another common replication that appears in large software repositories is method interface. It refers the return types, method names and parameter types of a method that repeats exactly or similarly across the code repositories.

If two methods contain the similar interfaces, it is very likely those perform same functionalities either entirely or at least partially. When those methods contain the same interface and perform similar functionalities, it indicates that these methods should be semantic or syntactic code clone to each other [2]. However, there is no evidence of whether such a hypothesis holds. To verify this to a certain level is very difficult, since various types of interface similarities can be possible. It is also challenging task to measure those similarities in different types of code clones (e.g., intra-project, inter-project, Type-1, Type-2 and Type-3). If it can be proved that interface similarity is significantly related to code clone, this can be helpful for clone detection, tracking and management.

Various clone detection techniques are proposed throughout the last decade. According to Roy et al. [3] clone detection techniques, using various representation, includes Textual [4], AST based [5], Token based [6] and Dependency graph based [7] etc. Although these tools are focused on clone detection, the effect and relationship of interfaces in code clones have never been analyzed. Several code search techniques such as Keyword Based Code Search (KBCS) [8] and Interface Driven Code Search (IDCS) [9] provide code searching with method interface information but never investigated the relationship between code clones and interfaces. A recent study [2] has discussed the effect of Interface Redundancy (IR) in code search but has not clearly explained the effect and impact of interface similarities in different types of code clones.

To explore the relationship and effects of interfaces similarity in code clones, an exploratory study is conducted on three subject systems (e.g., small, medium, and large). The subject systems contain 35, 109 and 24,558 open source Java projects crawled from SourgeForg, GitHub, Google Code etc. First, various types of method clone lists are detected by using two prominent clone detection tools SourcererCC [6] and NiCad [4]. After that, interface information (e.g., return types, method names, parameter types) is extracted for each method clone from its source code. For every interface, keywords and synonyms are also identified by extracting its method name. Finally, using this extracted information various types of interface similarities are found by satisfying similarity conditions. The percentage of how many clones satisfy each condition is measured. These results are used for establishing the desired relationship. More specifically, the aim of this study is to seek the answers of the following research questions.

RQ_1: *What does percentage of interface similarities occur in intra-project and inter-project method clones with various similarity combinations?*

RQ$_2$: *Are the intensities of interface similarity different in various types of clones and which clone-type(s) have higher possibilities to be detected by using interface similarity?*

RQ$_3$: *How does interface similarity relates to code clone detection? More specifically, how many code clones occur due to interface similarity?*

The experimental results show that 87.91% intra-project and 59.17% inter-project clones contain similar interfaces. Similarity of interfaces in Type-1,Type-2 and Type-3 clones are 100%, 83.47% and 81.90% respectively that infers interface similarity may have significant relationship with code clones. In future, it can be beneficial to design a new lightweight interface driven code clone detector.

The remainder of this paper is structured as follows. Section II presents background knowledge about the topics essential to understanding this study, such as code fragments, various types of clones [1]. Section III describes study design with the experimental dataset. Study results and discussion on these in the light of the research questions and the limitation of the results are represented in Section IV. Threats to validity are mentioned in Section V. Section VI deals with the existing works which are strongly relevant to the study. Finally, Section VII concludes the paper by summarizing the contribution and possible future direction of this study

II. Background

Subject System: Subject system refers to a code repository with different types of projects. These projects vary from one another by application domains and the number of files and Line of Code (LOC). Clone detection is applied on each project of the subject systems [1].

Code Fragment: It refers to a successive segment of source code. It is specified by the triple (*f, s, e*), where *f* represents the source file, *s* represents the line number from where the method starts on and the line at which it ends is represented by *e* [1].

Clone Pair/ Method Clone: Clone pair is a pair of code fragments that are similar or identical to each other. However, if two methods are cloned, it is specified as method clone. In this study, clone pair is considered as method clone. It is expressed by the triple (m_1, m_2, *t*), where m_1 and m_2 represent the similar methods, and the clone type is specified by *t* [1].

Intra-Project Clone: It refers to a clone pair where the methods are found in the same project within a subject system.

Inter-Project Clone: It occurs when a clone pair contains methods from two different projects within a subject system.

Type-1 Clone: Syntactically identical methods, except for differences in white-space, layout, and comments [1].

Type-2 Clone: Syntactically identical methods, except for differences in white-space, layout, comments, identifier names and literal values [1].

Type-3 Clone: Syntactically identical methods which differ at the statement level. The fragments have statements added, modified and/or removed with respect to each other, in addition to Type-1 and Type-2 clone differences [1].

III. Study Design

The aim of this study is to investigate the relationship, impact and effect of method interfaces in code clones. To do so, three types of subject systems are chosen (e.g., *Small, Medium* and *Large*) as the experimental dataset. Next, two accurate clone detection tools SourcererCC [6] and NiCad [4] are used to identify all the clones present in each type of subject system. As the granularity of clone detection was set to method level, these tools provide a list of method clones. For each clone, interface information are extracted from its source code for further statistical analysis to seek answers to the research questions mentioned in Section I. An overview of the study design is depicted in Figure 1.

A. Experimental Dataset Selection

Based on the number of projects and selection processes, experimental dataset of this study are classified as *Small*, *Medium* and *Large* subject systems.

1) Small Subject System (SSS): 35 open source Apache Java projects are selected as SSS. These projects are varied based on their size and functionalities including natural language processing libraries, network and database systems, etc. These projects are also used in several code clone studies [6]. A detailed description of these 35 projects can be found in this link[1].

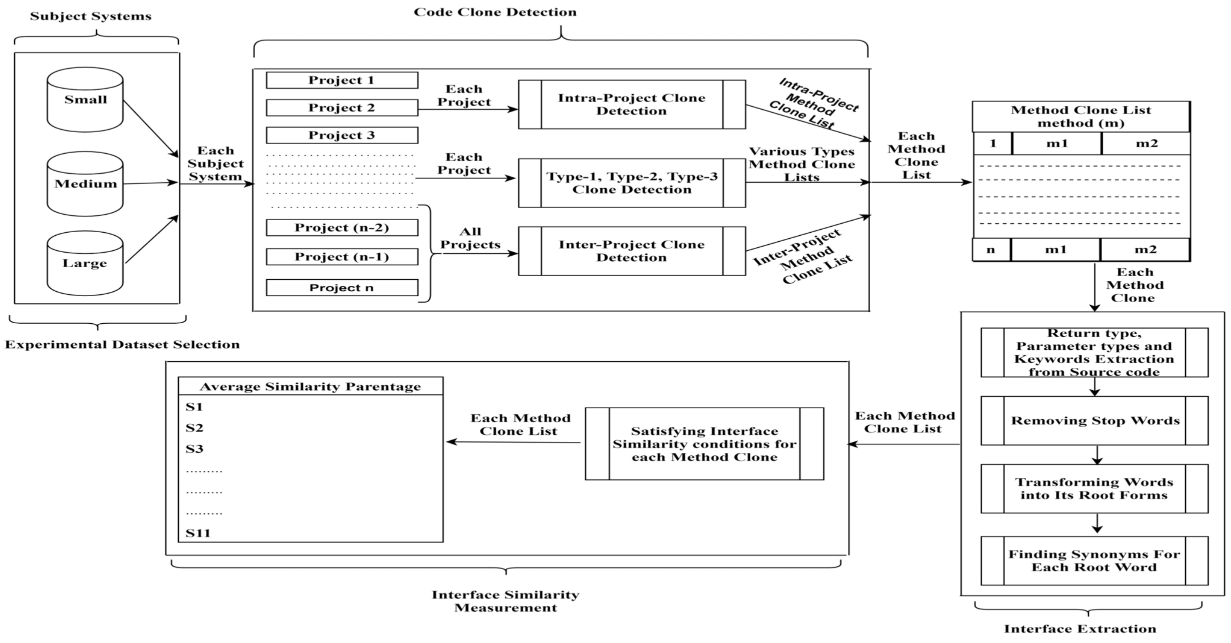
2) Medium Subject System (MSS): SF100 [10] is a statistically sound test data generation benchmark containing 100 open source Java projects. Later it has been extended by adding 10 most downloaded projects from SourceForge[2] called SF110 [11]. Here SF110 is used as MSS. These benchmark SF100 and SF110 were previously used in several studies [12], [9]. However, a project namely *Liferay Portal* is removed from SF110 because it contains 8,335 Java files and 1,552,597 LOC that create extreme value problem for statistical analysis.

3) Large Subject System (LSS): IJaDataset-2.0[3], a large Java source code repository, covers above 24k projects crawled from GitHub, SourceForge etc. While developing a clone detection benchmark BigCloneBench [13] and framework called BigCloneEval, Jeffrey et al [14] modified the IJaDataset-2.0. The modified version of IJaDataset-2.0 is used as LSS. TABLE I represents the summary of *Small, Medium and Large* subject systems.

B. Code Clone Detection

Two popular token-based clone detection tools SourcererCC [6] and NiCad [4] are used in this study because these tools achieve high recall and precision, in comparison to any other existing tools [2]. Since these tools give high recall with

[1]https://projects.apache.org/
[2]http://sourceforge.net
[3]IJaDataset 2.0, http://secold.org/ projects/seclone



Subject Systems
Small
Medium
Large
Experimental Dataset Selection
Each Subject System
Code Clone Detection
Project 1
Project 2
Project 3
Project (n-2)
Project (n-1)
Project n
Each Project
Each Project
All Projects
Intra-Project Clone Detection
Type-1, Type-2, Type-3 Clone Detection
Inter-Project Clone Detection
Intra-Project Method Clone List
Various Types Method Clone Lists
Inter-Project Method Clone List
Each Method Clone List
Method Clone List method (m)
1
m1
m2
n
m1
m2
Each Method Clone
Return type, Parameter types and Keywords Extraction from Source code
Removing Stop Words
Transforming Words into Its Root Forms
Finding Synonyms For Each Root Word
Interface Extraction
Each Method Clone List
Satisfying Interface Similarity conditions for each Method Clone
Each Method Clone List
Average Similarity Parentage
S1
S2
S3
S11
Interface Similarity Measurement

Figure 1. Overview of Study Design

TABLE I. SUMMARY OF SUBJECT SYSTEMS

Features	Subject System		
	Small	Medium	Large
Name	Apache	SF110	IJaDataset 2.0
Java Projects	35	109	24,558
Java Files	13,122	19,492	2,078,126
LOC	1,711,237	3,630,723	300,000,000

TABLE II. SUMMARY OF DETECTED CLONES

Clone Type	Small	Medium	Large	Total
Intra-Project	55,105	82,403	47,852	185,360
Inter-Project	2,352	20,342	17,265	39959
Type-1	9,099	4,826	47,567	61,492
Type-2	26,737	22,996	4,233	53,966
Type-3	37,185	29,875	6,659	73,717
Total	130,478	160,440	123,576	414,494

minimum 6 lines method level granularity and 80% similarity threshold, the same configuration is set for this study. SourcererCC [6] has been used for detecting intra-project and inter-project method clone detection. On the other hand, various types (e.g., Type-1, Type-2, and Type-3) of method clones are identified by NiCad [4]. Now, for each subject system, accumulating the result of the above mentioned steps, three types of clone lists are found. The intra-project method clones exist in the subject systems. All the inter-project method clones which are drawn from all projects of each subject system and different types of method clones (e.g; Type-1, Type-2 and Type-3) are present in each subject system. A list of detected clones is shown in TABLE II.

C. Interface Extraction

For every method clone, the source code of those methods are collected from their respective projects. After that, the source codes of each method are transformed into AST representation by using *Eclipse's ASTParser*[4]. Next, interface information such as return type, method name and parameter types are extracted by traversing AST nodes. In a method clone, each method name may contain single or multiple keywords that describe its functionality. These keywords are also extracted by following Java *CamelCase*[5] naming convention. Then keywords are transformed into its root form by removing stop words. It is performed by using a standard stop-word list[6]. Next, stemming is performed on those words. Stemming can be done based on various algorithms and tools. Here, *Apache Lucene*[7] Porter Stemmer is used. After that, synonyms are identified by using Java WordNet Library [15]. Finally, for each method clone pair return type, parameter types, keywords and a set of synonyms are retrieved.

Various types of interface similarity conditions are constructed with the combination of interface information. For example, one of the conditions is two interfaces are similar if their return types are similar. In total 11 types of interface similarity conditions are considered. Each conditions is represented by a unique identifier such as S_1 represents the condition of return type similarity of two methods in a method clone. A complete list of 11 types of similarity conditions is explained in TABLE III.

D. Interface Similarity Measurement

The percentage of how many clones satisfy each similarity condition is calculated since these helps to normalize the

[4]https://github.com/eclipse/eclipse.jdt.core
[5]https://docs.oracle.com/javase/tutorial/java/javaOO/methods.html
[6]http://www.ranks.nl/stopwords
[7]https://lucene.apache.org/core/

TABLE III. LIST OF INTERFACE SIMILARITY CONDITIONS

Identifier	Similarity Conditions
S_1	Return types are similar
S_2	Number and types of parameters are similar
S_3	At least one parameter is similar
S_4	Return types and parameter types are similar
S_5	Return types and at least one parameter type are similar
S_6	At least one keyword extracted from method name is similar
S_7	Keywords extracted from method name are similar
S_8	At least one synonym of extracted keyword is similar
S_9	At least one synonym from all keywords are similar
S_{10}	Return types and all keywords and parameters are similar
S_{11}	Return and parameter types, at least one keyword are similar

results value. For example, in SSS for a single project, detected clone lists (e.g., intra-project, Type-1, Type-2 and Type-3 clone list) are taken. Then, for each clone list, how many clones satisfy each similarity condition are counted and obtain the percentage for each type of similarity condition for that project. After that, by averaging all project's percentage, the overall mean is calculated for the subject system. For example, from TABLE IV it is observed that on average 91.73% intra-project clone satisfy S_1 similarity condition within SSS. Similarly, following the same process, for each type of similarity condition, percentage and overall mean are calculated in MSS and LSS. However, it is noted that the number and types of parameters received by a method are more important than the order in which those have appeared. So, while satisfying the similarity conditions associated to parameters, the ordering of those parameters are ignored. Here, interface `int read(String, int, int)` and `int read(int, int, String)`are considered to be similar based on the number of parameters and types ignoring the ordering of those parameters.

IV. STUDY RESULTS

In this section, the experimental result analysis of code clone and its relationship with interfaces are presented. The results are found by satisfying 11 similarity conditions (e.g., S_1,S_2...,S_{11}). So, the answers of those research questions (mentioned in Section I) are drawn based on the fulfilling of the similarity conditions.

***RQ$_1$*: What does percentage of interface similarities occur in intra-project and inter-project method clones with various similarity combinations?**

To answer this question, the results are presented with respect to intra-project and inter-project method clones. The numbers of intra-project clones found in SSS, MSS and LSS are 55,105 82,403 and 47,852 respectively that amounts in total 185,360 method clones. In the SSS, 35 projects contain a considerable number of clones. However, in MSS and LSS only 96 and 1,841 projects include method clones. The main reason for this is, most of the method code fragments do not satisfy 80% similarity threshold and minimum 6 lines configuration parameters. On the other hand, the number of detected inter-project clones in SSS, MSS and LSS are 2,352, 20,342 and 17,265 respectively. It amounts in total 39,959 inter-project method clones. The percentage of interface similarities in intra-project and inter-project clones in each subject system is shown in TABLE IV and TABLE V.

TABLE IV. INTERFACE SIMILARITIES IN INTRA-PROJECT CLONES

Conditions	Subject System			Average(%)
	Small	Medium	Large	
S_1	87.14%	88.78%	99.28%	91.73
S_2	84.92%	87.97%	98.68%	90.52
S_3	92.34%	91.55%	98.90%	94.26
S_4	74.12%	78.34%	98.14%	83.53
S_5	80.85%	81%	98.28%	86.71
S_6	84.45%	80.44%	98.84%	87.91
S_7	70.22%	60.84%	97.37%	76.14
S_8	88.39%	85.45%	98.96%	90.93
S_9	65.06%	54.47%	97.13%	72.22
S_{10}	70.22%	60.84%	97.37%	76.14
S_{11}	84.45%	80.44%	98.84%	87.91

TABLE V. INTERFACE SIMILARITIES IN INTER-PROJECT CLONES

Conditions	Subject System			Average(%)
	Small	Medium	Large	
S_1	97.49%	76.05%	94.06%	89.20
S_2	92.60%	99.90%	91.58%	94.69
S_3	94.22%	99.80%	91.37%	95.13
S_4	91.20%	76.01%	86.88%	84.69
S_5	92.73%	76.02%	87.58%	85.44
S_6	59.52%	29.07%	88.93%	59.17
S_7	53.19%	38.21%	68.13%	53.18
S_8	88.90%	83.80%	87.01%	86.57
S_9	51.91%	32.43%	63.35%	49.23
S_{10}	53.19%	38.21%	68.13%	53.18
S_{11}	59.52%	29.07%	88.93%	59.17

It is observed that the average rate of fulfilling each type of similarity condition is almost consistent in all subject systems. It is an indication that interface similarity is related to method clone. Especially, on average above 85% of the clones satisfy only return type and parameter types based similarity conditions such as S_1, S_2, S_3, S_4 and S_5. The is besause if two methods contain similar return types and parameters, it means those take similar input and provide similar output. Most of the time these methods are considered to be clones as those perform similar functionality omitting the keyword those contain in method names [16]. It infers that approximately above 85% intra-project and inter-project method clones contain similar return type and parameter types. As keywords have less significant influences on method functionality comparing to its return type and parameter types, only 76.14% and 87.91% intra-project clones satisfy conditions S_{10} and S_{11}. In comparing to intra-project clones, similarity conditions S_{10} and S_{11} are not satisfied by a significant number of inter-project clones. Since the usage of inappropriate naming convention, improper keywords and generic type prevents method clones satisfying similarity conditions S_{10} and S_{11}. Only 53.18% and 59.17% intra-project clones meet S_{10} and S_{11} conditions respectively that infers 59.17% inter-project clone contains similar keywords from method names, return type and parameter types.

From TABLE IV and TABLE V, it is observed that conditions associated to keyword and synonym such as S_7, S_8,

S_9, S_{10}, and S_{11} are not satisfied by the significant number of clones. Besides, conditions associated to return type and parameter types (e.g., S_1, S_2, S_3 and S_4), are not satisfied by almost 15-20% clones. To find the reasons, the source code of those clones are manually inspected. The first reason is usage of inappropriate naming convention. In some clones, one of the method names is not written by following CamelCase naming convention. Some clones contain PascalCase and Snake case or underscore case naming convention. While splitting method name by following CamelCase naming convention, keywords cannot be extracted. As a result, clones are failed to fulfill these (e.g., S_7, S_8, S_9 and S_{11}) conditions. Another reason is usage of improper keywords in method names. For example, in a method clone, two interfaces are `List<Integer> sort(List<Integer>)` and `List<Integer> x(List<Integer>)`. In the second interface, developers use improper term in the method name that does not describe its functionality. So, correct keywords cannot be extracted from its method name that prevents those clones satisfying conditions (e.g., S_7, S_8, S_9, S_{10} and S_{11}) having similar return and parameter types.

The second reason is type mismatch problem. In some clones, the return type and parameter types are not exactly the same, but based on programming knowledge, those are considered to be similar. For example, in a method clone, two interfaces are `int[] bubbleSort(int[])` and `double[] bubbleSort(double[])`. These two methods perform same functionality on two different type of data (e.g., int and double). These methods also contain the similar syntactic implementation in the method body. However, in this study, these interfaces are considered to be dissimilar since return types and parameter types are not exactly matched. Type mismatch problems have also occurred in some clones for the usages of generic return type and parameter type. For example type mismatch occurs between interface `<T extends Number> void copyList (List<T>dest, List<T> src)` and `List<Integer> copyList (List<Integer>src, List<Integer>dest)`. As a result, a significant number of clones (both intra-project and inter-project clones) does not satisfy those similarity conditions adequately.

***RQ$_2$*: Are the intensities of interface similarity different in various types of clones and which clone-type(s) have higher possibilities to be detected by using interface similarity?**

This research question is answered with respect to Type-1, Type-2 and Type-3 method clones. The total numbers of detected Type-1, Type-2 and Type-3 method clones in three subject systems are 61,492, 53,966 and 73,717 respectively. TABLE VI, TABLE VII and TABLE VIII represent the summary of satisfying interface similarities by Type-1, Type-2 and Type-3 clones respectively. From TABLE VI, it is observed that conditions (i.e., S_1, S_2,...,S_{11}) satisfied by Type-1 clones are 100%. It is consistent with Type-1 method clone definition.

TABLE VI. INTERFACE SIMILARITIES IN TYPE-1 CLONES

Conditions	Subject System			Average(%)
	Small	Medium	Large	
S_1	100%	100%	100%	100
S_2	100%	100%	100%	100
S_3	100%	100%	100%	100
S_4	100%	100%	100%	100
S_5	100%	100%	100%	100
S_6	100%	100%	100%	100
S_7	100%	100%	100%	100
S_8	100%	100%	100%	100
S_9	99.94%	100%	100%	99.98
S_{10}	100%	100%	100%	100
S_{11}	100%	100%	100%	100

TABLE VII. INTERFACE SIMILARITIES IN TYPE-2 CLONES

Conditions	Subject System			Average(%)
	Small	Medium	Large	
S_1	83.26%	81.81%	98.72%	87.93
S_2	83.98%	81.91%	97.68%	87.86
S_3	87.30%	87.34%	98.02%	90.88
S_4	69.39%	65.23%	96.55%	77.06
S_5	71.79%	69.46%	96.88%	79.38
S_6	81.39%	80.40%	88.61%	83.47
S_7	43.47%	49.49%	19.23%	37.40
S_8	81.57%	76.68%	94.05%	84.10
S_9	37.89%	40.86%	17.06%	31.94
S_{10}	43.47%	49.49%	19.23%	37.40
S_{11}	81.39%	80.40%	88.61%	83.47

Since syntactical modifications do not occur in Type-1 method clone, interfaces of those clones always remain similar.

From TABLE VII and TABLE VIII, it is found that above 80% Type-2 and Type-3 method clones satisfy return type and parameter types based similarity conditions (e.g., S_1, S_2, S_3 and S_4). However, a significant number of Type-2 and Type-3 clones satisfy conditions S_{10} and S_{11}. On average 83.47% Type-2 and 81.90% Type-3 clones fulfill S_{10} and S_{11} conditions respectively. As renaming of identifiers and literal values occur in Type-2 clones and some addition or deletion of statement occur in Type-3 method clones, some method clones fail to fulfill some similarity conditions. For example, in a Type-2 method clone interfaces are `List<String> getContent(File)` and `ArrayList<String> getContent(File)` respectively. However, by following Java polymorphism features, these two methods contain similar interfaces, but here these are considered dissimilar. This is because that exact matching is performed to measure the return type similarity. Besides, inappropriate naming convention, type mismatch problem and usage of generic type prevent on average 20% Type-2 and Type-3 clones satisfying similarity conditions. It infers that approximately 83.47% Type-2 and 81.90% Type-3 clones contain similar keywords from method name, return and parameter types. So, the intensity of interface similarity is higher in Type-1 compared to Type-2 and Type-3 clones.

From the experimental result, it has been analyzed that all types of method clones can be detected using interface similarities. There is 100% probability that Type-1 method clones can be identified by performing exact interface information

TABLE VIII. INTERFACE SIMILARITIES IN TYPE-3 CLONES

Conditions	Subject System			Average(%)
	Small	Medium	Large	
S_1	84.72%	85.68%	91.44%	87.28
S_2	79.07%	83.13%	87.90%	83.37
S_3	85.12%	85.93%	88.21%	86.42
S_4	66.84%	71.75%	81.09%	73.23
S_5	72.23%	73.84%	81.95%	76.01
S_6	80.94%	76.87%	87.90%	81.90
S_7	60.29%	58.39%	77.51%	65.40
S_8	82.40%	80.32%	84.35%	82.36
S_9	55.41%	51.61%	71.75%	59.59
S_{10}	60.29%	58.39%	77.51%	65.40
S_{11}	80.94%	76.87%	87.90%	81.90

matching. Above 80% Type-2 and Type-3 method clones can be detected by interface similarities since 83.47% Type-2 and 81.90% Type-3 clones contain similar interfaces. However, while detecting Type-2 and Type-3 clones, more than 80% clone can be detected by incorporating to the inappropriate naming convention, type mismatch problem and usage of the generic types.

RQ$_3$*: *How does interface similarity relates to code clone detection? More specifically, how many code clones occur due to interface similarity?

In this question, the relationship between classical code clone and interface similarity has been investigated. The main question is how much of clones occurs due to interface similarity. In this case, only the intra-project clones are considered because these are the method clones that are implemented by the developers of each project. Besides, for interface similarity measurement, only those clones are considered that satisfy similarity condition S_{11}. This is because it ensures both methods in the each clone pair contain same return types, at least one keyword from method name and one parameter type. TABLE IX represents the number of intra-project method clones in each subject system. It also provides both the numbers of clones that satisfy and do not satisfy S_{11} similarity condition. It is found that out of 1,85,360 intra-project method clones only 25,241 clones do not contain similar interfaces that refer only 13.62% clones do not satisfy S_{11} similarity condition. Because of the inappropriate naming convention, type mismatch and generic type matching problem, these clones fail to satisfy similarity condition that is discussed while answering research questions **RQ$_1$** and **RQ$_2$**. On the other hand, it is observed that in total 1,60,119 method clones contain similar interface. It infers that 86.38% clones occur due to interface similarity. It is a very important result since it is the evidence that interface similarity may have significant relationship to classical method clone detection.

V. Threats to Validity

The number of detected clones may increase or decrease with the variation of detection parameters used by SourcererCC [6] and NiCad [4]. In an exhaustive study, Wang et al. [17] observed that clone detection tools are affected by confounding configuration parameter choice problem. So, SourcererCC [6] and NiCad [4] are also affected by this problem. However, here standard configuration settings (e.g., 80% similarity with minimum 6 lines) are used by SourcererCC [6] and NiCad [4] that prevents both tools getting false positive clones and ignoring the `getter`, `setter`, and abstract methods of Java class attributes.

In this study, NiCad has been used for detecting Type-1, Type-2 and Type-3 clones. So, any other clone detection tools can provide different experimental results. Svajlenko et al. [18] has shown that for detecting three types of clone, NiCad is very accurate in comparison with other clone detection tools. Similarly, in some previous clone detection studies [6], it is observed that SourcererCC is also very accurate for detecting intra-project and inter-project clones in large code repositories. So, the clones detected by SourcererCC and NicCad have significant impact on the results of this study.

The experimental results may vary if the subject systems are changed. However, subject systems used in this experiment are enough to take a complete decision on the relationship and effects of interface similarities in code clones. The reason is that the candidate systems differs in terms of the application domains, size, revision and the presences of various types clones. These subject systems contain 35, 109 and 24,558 open source Java Projects extracted from SourceForge, Git-Hub, Google Code etc. So, for instance, these projects can be representative of a small, medium and large company's local repositories. These are also used in many code clone studies [6]. The experimental results implied from this subject system should be statistically sound.

TABLE IX. INTRA-PROJECT CLONES SATISFYING CONDITION S_{11}

Clones	Subject System			
	Small	Medium	Large	**Total**
Intra Project	55,105	82,403	47,852	**1,85,360**
Satisfy S_{11}	46,537	66,285	47,297	**1,60,119**
Percentage (%)	84.45	80.44	98.84	**86.38%**
Do not Satisfy S_{11}	8,568	16,118	555	**25,241**
Percentage (%)	15.55	19.56	1.16	**13.62%**

Reproducibility: All the necessary artifacts of this study are publicly available[8]. For generating the statistical results to verify the claims of this study, these artifacts include source code of subject systems, all types of detected clones e.g., intra-project, inter-project, Type-1, Type-2 ad Type-3), raw data generated in the study, and the source code of analyzing interface similarity[9].

VI. Related Work

The study of establishing the relationship between method interfaces and code clones are closely related with various areas (e.g., Clone Detection, Code Search etc.) of software engineering research that have similar concepts. A brief description of those areas is mentioned below.

[8]https://github.com/MisuBeImp/APSEC-2017-Paper-Artifacts
[9]https://github.com/MisuBeImp/CloneInterfaceSimilarityDetector

A. Clone Detection

Numbers of clone detection approaches have been proposed in the literature. Based on algorithm and source code representation, these approaches differ from each other. Roy et al. have performed a comprehensive survey focusing the strength and limitation of various clone detection approaches [3]. According to that clone detection techniques can be categorized into various types such String-based, Token-based, Tree-based techniques [3]. String-based techniques use the source code with little or no transformation and use string matching algorithms. For example, Marcus et al. used the Latent Semantic Indexing (LSI) algorithm for detecting high-level concept clones [19]. Similarly, Cordy et al. also applied the Longest Common Subsequence (LCS) algorithm in their tool called NiCard [4] for an efficient text line comparison to find nearly miss matched clones. Since, string matching algorithms do not work faster for larger dataset, these techniques face scalability issues.

Tree-based techniques detect clones by finding similar subtrees. The source code are parsed into its AST representation. Various graph or tree matching algorithms are used to detect similar subtrees. Jiang et al. proposed a novel approach for matching similar trees in their tool Deckard [5]. They computed a certain characteristic vector from the AST and used Locality Sensitive Hashing (LSH) to cluster similar vectors based on Euclidean distance metric for detecting clones. However, AST based techniques face scalability issues as parse trees contain a lot of information consuming high memory.

Token based techniques transform the source code into a sequence of the lexical token with a predefined granularity level. Next, the comparison is done between the tokens. The transformation of the token is done by using the lexical analyzer to parse the source code, and rule-based transformation is applied to generate a stream of the token. Sajnani et al has introduced a token based approach called SourcererCC [6] using sub block overlapping heuristic and token positioning filtering for efficient comparison [6]. However, these techniques only concerned in clone detection neither focused on interface similarity nor investigated the effects of interface similarities in code clones.

B. Code Search

Code search has become popular with the increasing movement of open source code. Existing code search techniques includes Keyword Based Code Search (KBCS) [8], Semantic Based Code Search (SBCS) [20], and Test Driven Code Search (TDCS) [21]. Moreover, to extend code search techniques, researchers have proposed a sophisticated approach called Interface Driven Code Search (IDCS) [22]. It allows users to search code in code libraries, by using interface information. IDCS first crawls all the methods from the code libraries and extracts interfaces for indexing similar methods. It can be performed by using existing code search tools such as Sourcerer [23]. However, while indexing, IDCS cannot select proper terms for similar codes with the analogous functionality. Recently a study [16] has improved IDCS performance by indexing similar methods under appropriate terms. The study has shown that if two methods contain similar return types and parameters, most of the time those perform similar functionality omitting the keyword they contain in method names [24]. The findings also support this interface similarity study, as the result analysis shows that average interface similarity of method clones based on return type and parameters are better than those similarities which are based on keywords (i.g,.root words and synonyms). However, IDCS differs from interface similarity study since it only finds relevant code based on interface not the method clones.

C. Interface Redundancy

Interface Redundancy (IR) represents the repetition of whole method interface (e.g.,return type, method name, and parameters types) across the software corpus. Paula et al. have first introduced it in an exploratory study [2]. Their study is much focused on exploring the impacts and effects of IR in code search. They extracted 380,000 methods from a code corpus (i.g.,a medium subject system SF100) in a relational database by Sourcerer [23]. To identify redundant interfaces, they have performed IDCS on the databse in two step. First, IDCS is performed without query expansion. Then, again it is performed with Automatic Query Expansion (AQE) [25] to overcome the keyword related vocabulary mismatch and different thesauri [12] (e.g., types of lists contained within the Java API) problem. The results has shown that 80% project of the targeted repositories contain redundant interfaces. Besides, it is found that the knowledge of redundant interface in code repositories improves the performance of code search especially in IDCS. Additionally, it is observed that IR has diverged from traditional code cloning since in their study only 0.002% IR is related to method clones [2].

VII. Conclusion

The relationship between method clones and method interfaces have never been studied before. In this study, an exploratory study has been performed to investigate the relationship, impact and effects of interfaces in code clones. In the first step, three types of subject systems (*Small*, *Medium* and *Large*) are selected as the experimental dataset that contain 35, 109 and 24,558 open source Java projects respectively. Next, two token based clone detection tools such as SourcererCC [6] and NiCad [4] are used to identify all types of clones (e.g., intra-project, inter-project, Type-1, Type-2 ad Type-3) within the subject systems. The average number of clones found are 130,478, 160,440 and 123,576 respectively. After that, interface information (e.g return type, method name and parameter types) of those clones are extracted from the source code. Finally, interfaces similarity is measured in method clones by satisfying similarity conditions.

The experimental result analysis shows that on average 87.91% intra-project clones and 59.17% inter-project clones contain similar interface (i.e., return types and at least one root word and at least one parameter types are similar). Besides, 100% Type-1, 83.47% Type-2 and 81.90% Type-3 clone contain similar interfaces. However, use of inappropriate naming convention, generic type and type mismatch problem prevent some clones satisfying interface similarities which ushers new research directions. Besides, the findings help to design new interface driven code clone detection tool.

References

[1] C. K. Roy and J. R. Cordy, "A survey on software clone detection research," *Queens School of Computing TR*, vol. 541, no. 115, pp. 64–68, 2007.

[2] A. C. de Paula, E. Guerra, C. V. Lopes, H. Sajnani, and O. A. L. Lemos, "An exploratory study of interface redundancy in code repositories," in *Source Code Analysis and Manipulation (SCAM), 2016 IEEE 16th International Working Conference on*. IEEE, 2016, pp. 107–116.

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

[4] J. R. Cordy and C. K. Roy, "The nicad clone detector," in *Program Comprehension (ICPC), 2011 IEEE 19th International Conference on*. IEEE, 2011, pp. 219–220.

[5] L. Jiang, G. Misherghi, Z. Su, and S. Glondu, "Deckard: Scalable and accurate tree-based detection of code clones," in *Proceedings of the 29th international conference on Software Engineering*. IEEE Computer Society, 2007, pp. 96–105.

[6] H. Sajnani, V. Saini, J. Svajlenko, C. K. Roy, and C. V. Lopes, "Sourcerercc: scaling code clone detection to big-code," in *Proceedings of the 38th International Conference on Software Engineering*. ACM, 2016, pp. 1157–1168.

[7] R. Komondoor and S. Horwitz, "Effective, automatic procedure extraction," in *Program Comprehension, 2003. 11th IEEE International Workshop on*. IEEE, 2003, pp. 33–42.

[8] W. B. Frakes and B. A. Nejmeh, "Software reuse through information retrieval," in *ACM SIGIR Forum*, vol. 21, no. 1-2. ACM, 1986, pp. 30–36.

[9] O. A. L. Lemos, A. C. de Paula, H. Sajnani, and C. V. Lopes, "Can the use of types and query expansion help improve large-scale code search?" in *Source Code Analysis and Manipulation (SCAM), 2015 IEEE 15th International Working Conference on*. IEEE, 2015, pp. 41–50.

[10] G. Fraser and A. Arcuri, "Sound empirical evidence in software testing," in *Software Engineering (ICSE), 2012 34th International Conference on*. IEEE, 2012, pp. 178–188.

[11] ——, "A large-scale evaluation of automated unit test generation using evosuite," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 24, no. 2, p. 8, 2014.

[12] O. A. Lemos, A. C. de Paula, F. C. Zanichelli, and C. V. Lopes, "Thesaurus-based automatic query expansion for interface-driven code search," in *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 2014, pp. 212–221.

[13] J. Svajlenko, J. F. Islam, I. Keivanloo, C. K. Roy, and M. M. Mia, "Towards a big data curated benchmark of inter-project code clones," in *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*. IEEE, 2014, pp. 476–480.

[14] J. Svajlenko and C. K. Roy, "Bigcloneeval: A clone detection tool evaluation framework with bigclonebench," in *Software Maintenance and Evolution (ICSME), 2016 IEEE International Conference on*. IEEE, 2016, pp. 596–600.

[15] M. A. Finlayson, "Java libraries for accessing the princeton wordnet: Comparison and evaluation," in *Proceedings of the 7th Global Wordnet Conference, Tartu, Estonia*, vol. 137, 2014.

[16] A. Satter and K. Sakib, "Improving recall in code search by indexing similar codes under proper terms." in *QuASoQ/TDA@ APSEC*, 2016, pp. 35–42.

[17] T. Wang, M. Harman, Y. Jia, and J. Krinke, "Searching for better configurations: a rigorous approach to clone evaluation," in *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. ACM, 2013, pp. 455–465.

[18] J. Svajlenko and C. K. Roy, "Evaluating modern clone detection tools," in *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*. IEEE, 2014, pp. 321–330.

[19] A. Marcus and J. I. Maletic, "Identification of high-level concept clones in source code," in *Automated Software Engineering, 2001.(ASE 2001). Proceedings. 16th Annual International Conference on*. IEEE, 2001, pp. 107–114.

[20] S. P. Reiss, "Semantics-based code search," in *Proceedings of the 31st International Conference on Software Engineering*. IEEE Computer Society, 2009, pp. 243–253.

[21] O. A. Lazzarini Lemos, S. K. Bajracharya, and J. Ossher, "Codegenie:: a tool for test-driven source code search," in *Companion to the 22nd ACM SIGPLAN conference on Object-oriented programming systems and applications companion*. ACM, 2007, pp. 917–918.

[22] A. M. Zaremski and J. M. Wing, "Signature matching: a tool for using software libraries," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 4, no. 2, pp. 146–170, 1995.

[23] S. Bajracharya, J. Ossher, and C. Lopes, "Sourcerer: An infrastructure for large-scale collection and analysis of open-source code," *Science of Computer Programming*, vol. 79, pp. 241–259, 2014.

[24] A. Satter and K. Sakib, "A similarity-based method retrieval technique to improve effectiveness in code search," in *Companion to the first International Conference on the Art, Science and Engineering of Programming*. ACM, 2017, p. 39.

[25] C. Carpineto and G. Romano, "A survey of automatic query expansion in information retrieval," *ACM Computing Surveys (CSUR)*, vol. 44, no. 1, p. 1, 2012.