On the Feasibility of Detecting Cross-Platform Code Clones via Identifier Similarity

Xiao Cheng[1], Lingxiao Jiang[2], Hao Zhong[1], Haibo Yu[3], Jianjun Zhao[4]
[1]Department of Computer Science and Engineering, Shanghai Jiao Tong University, China
[2]School of Information Systems, Singapore Management University, Singapore
[3]School of Software, Shanghai Jiao Tong University, China
[4]Department of Advanced Information Technology, Kyushu University, Japan
{x.cheng, zhonghao, haibo_yu}@sjtu.edu.cn, lxjiang@smu.edu.sg, zhao@ait.kyushu-u.ac.jp

ABSTRACT

More and more mobile applications run on multiple mobile operating systems to attract more users of different platforms. Although versions on different platforms are implemented in different programming languages (*e.g.*, Java and Objective-C), there must be many code snippets that implement the similar business logic on different platforms. Such code snippets are called cross-platform clones. It is challenging but essential to detect such clones for software maintenance. Due to the practice that developers usually use some common identifiers when implementing the same business logic on different platforms, in this paper, we investigate the identifier similarity of the same mobile application on different platforms and provide insights about the feasibility of cross-platform clone detection via identifier similarity. In our experiment, we have analyzed the source code of 18 open-source cross-platform applications which are implemented on Android, iOS and Windows Phone, and find that the smaller KL-Divergence the application has, the more accurate the clones detected by identifiers will be.

CCS Concepts

•**Software and its engineering** → *Software libraries and repositories;*

Keywords

cross-platform application; identifier similarity; code clone;

1. INTRODUCTION

Nowadays, mobile phones play a more and more important role in our daily life. To meet our increasing demands, almost all the mobile operating systems (*e.g.*, Android, iOS, and Windows Phone) provide similar built-in features such as Global Positioning System (GPS), Camera, Music, making calls and sending short messages. Due to the uniqueness of mobile operating systems together with the preference of users, intensive competition among them has been going on, ever since they were released. In order to attract more different users, one application is usually implemented on different mobile operating systems.

The application (app), which runs on multiple mobile operating systems, is called a *cross-platform application* [4]. For example, the *Google Map* app has both an Android and an iOS version with almost identical functionalities. To implement similar functionalities, there must be many code snippets that implement the identical or similar functionality. We call such code snippets as *cross-platform clones*. In order to maintain the same functionality, programmers have to locate and synchronize clones. For example, when developers modify a code snippet on a platform (*e.g.*, Android), all the clone instances in the other platforms (*e.g.*, iOS, and Windows Phone) may require the similar modifications to maintain the identical functionality. Due to the differences of architectures and programming languages, the maintenance is tedious and error-prone.

In literature, various clone detection approaches [6, 7, 8, 9, 10, 14, 2] have been proposed, which include the ones for the desktop project and the ones for Android app, but they only target at one mobile platform and focus on code clones in one programming language. Recently, researchers [11, 1] have proposed approaches to detecting cross-language clones for the .NET language family, which share a common intermediate language. However, Android apps which are implemented in Java, iOS apps which are implemented in Objective-C and Windows Phone apps which are implemented in C# do not share any common intermediate language or representation. It is challenging to detect such kind of cross-platform clones.

As functionalities on one platform can be used as a reference for implementation on another platform, we hypothesize that developers may use common identifiers when they implement similar functionalities on different platforms. Indeed, Zhong *et al.* [13] show that the similarity is adequate to detect many code mappings between Java and C#. However, it is still largely unknown whether the hypothesis holds in mobile apps since mobile platforms are more different than Java and C#. To fully explore the hypothesis, in this paper, we analyzed the source files of 18 mobile apps and 5 desktop projects. The mobile apps are implemented in Java, Objective-C, and C#. The desktop projects are implemented in Java and C#. Our research questions are as follows:

- **RQ1.** What are the distributions of identifier frequency in cross-platform apps?

SoftwareMining'16, September 3, 2016, Singapore, Singapore
© 2016 ACM. 978-1-4503-4511-8/16/09...$15.00
http://dx.doi.org/10.1145/2975961.2975967

Table 1: Subject Mobile Applications

Application	Android		iOS		WinPhone	
	#Files	LOC	#Files	LOC	#Files	LOC
BlueSenseNetworks	27	1,543	57	3,956	-	-
Cordova	60	5,350	41	3,576	-	-
CycleStreets	187	18,676	332	47,459	-	-
Digipost	104	11,198	74	9,597	-	-
DuckDuckGo	219	14,091	87	10,782	-	-
Facebook	305	42,665	252	32,008	-	-
IRCCloud	130	35,319	89	40,319	-	-
OwnTracks	92	10,674	74	13,949	-	-
RioBus	13	675	14	1,400	-	-
Adsota	4	349	3	93	16	708
BlinkID	43	4,306	18	963	13	815
LowaCodeCamp	18	725	22	1,355	23	705
OneSignal	89	7,491	11	1,613	22	1,235
OpenPKW	48	3,125	49	1,344	88	5,853
OwnCloud	223	33,764	199	42,349	95	7,114
VK	114	10,175	60	5,388	85	5,941
Wit	11	661	15	1,715	9	554
WordPress	590	89,642	362	62,567	137	21,769

- **RQ2.** How similar are the identifiers in cross-platform apps?
- **RQ3.** Is it feasibility to detect cross-platform clones via identifier similarity?

2. METHODOLOGY

To investigate our research questions, we normalize the source code of versions on different platforms and calculate the similarity between normalized code. To obtain an intuitive estimation of the similarity, we compare the similarity with desktop cross-platform projects, in which the similarity between identifiers is effective to detect code clones.

2.1 Normalizing

Normalizing is a process to remove uninteresting content (*e.g.*, comments) from the source code, and to transform the remaining content into normalized comparison units. In particular, the source code of each version is tokenized to a token stream, with the following steps:

- In order to relieve the impact of the difference between the comments which written in natural language, the comments are removed.
- The remaining code is tokenized as a token stream.
- Punctuations and numbers are removed from the token streams since they seldom indicate semantics.
- Camel case tokens are split by uppercase letters and tokens with underscores are split by the underscore. All tokens are then transformed to lowercase.

After normalization, the source code of the app on each platform will be transformed into a token stream.

2.2 Calculating Similarity

Bag of Words (BOW) [5] is a text representation technique widely used in Natural Language Processing. A text is represented as a bag of its words, disregarding the grammar and even the order of words. We adopt the BOW model to represent each token stream. For each bag of tokens, a characteristic vector $V(v_1, v_2, \ldots, v_n)$ is built, whose each dimension denotes the number of each token in the token stream.

Based on the characteristic vector, we calculate the distribution of the tokens $D(d_1, d_2, \ldots, d_n)$, where $d_i = \frac{v_i}{\sum_{j=1}^{n} v_j}$.

We utilize the similarity of the token distributions to reflect the identifier similarity of the source code. Kullback-Leibler divergence (KL-Divergence) [12] is a measure of the difference between probability distributions P and Q. For discrete probability distributions P and Q, the KL-Divergence of Q from P is defined as:

$$KLD(P \mid\mid Q) = \sum_{i=1}^{n} P(i) \log \frac{P(i)}{Q(i)} \quad (1)$$

We use KL-Divergence of the token distributions to measure the distance between the source code for two platforms (*e.g.*, Android and iOS). The larger the distance is, the less similar the identifiers are. Since the two distributions are symmetrical, for two distributions $D_i(d_{i1}, d_{i2}, \ldots, d_{in})$ and $D_j(d_{j1}, d_{j2}, \ldots, d_{jn})$, the distance is defined as:

$$Distance(D_i, D_j) = KLD(D_i \mid\mid D_j) + KLD(D_j \mid\mid D_i) \quad (2)$$

In order to avoid the appearance of value zero in the distribution vectors, which can trigger math error, for each pair of characteristic vectors, we replace zeros with ones.

Table 2: Subject Desktop Projects

Project		#Files	LOC	Distance
ANTLR3	Java	276	49,503	0.252
	C#	623	109,058	
FpML	Java	130	17,810	0.581
	C#	130	16,548	
Log4j/Log4net	Java	309	30,278	0.917
	C#	335	30,884	
Lucence	Java	6,098	862,288	0.489
	C#	2,929	434,306	
Spring	Java	6,308	548,782	0.725
	C#	2,717	223,894	

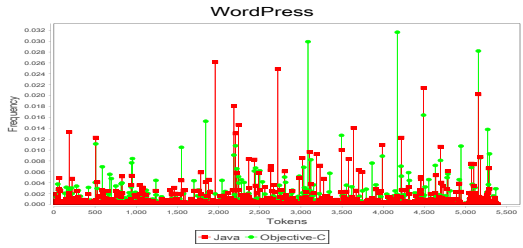
2.3 Comparison

The distance between the token distribution of apps on different platforms can reflect the identifier similarity. However, it is still not enough to answer **RQ3**. In order to find the relationship between identifier similarity and code clones to obtain intuitive results, we compare the token distribution distances of mobile apps with the distribution distances of desktop projects. In our previous work [3], we have detected cross-language clones through comparing revision histories that are recorded in repository logs, which is an identifier-similarity-based approach. This approach is mainly for desktop apps developed in more similar programming languages (*i.e.* Java and C#). It needs assessment whether similar techniques can be applied to mobile apps. Since it is time-consuming to apply the technique to mobile apps, in this paper, we compare the token distribution distance of mobile apps, which implemented in more different languages, with that of the desktop one, which implemented in more similar languages. Through this comparison, we estimate how clones can be detected from the cross-platform apps based on identifier similarity.

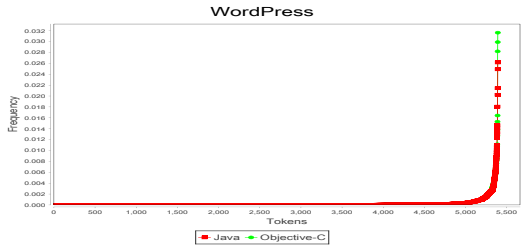
3. EMPIRICAL STUDY RESULTS

We collected 18 open-source cross-platform mobile apps from GitHub[1], 9 of which have Android and iOS versions and the other 9 of which have Android, iOS, and Windows Phone versions. Table 1 lists the basic statistics of our subjects. Columns "#Files" of Android, iOS, and Windows Phone platform denote the number of source code files. Columns "LOC" denotes the lines of code in these files. We collected 5 open source desktop cross-language projects as the control group. Table 2 lists the basic statistics of them.

[1]http://www.github.com

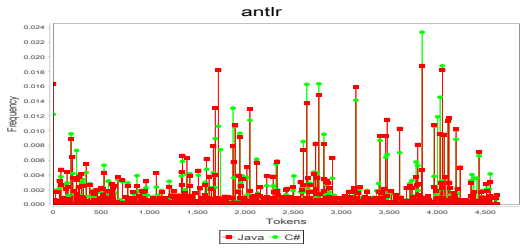


(a) Token Distribution

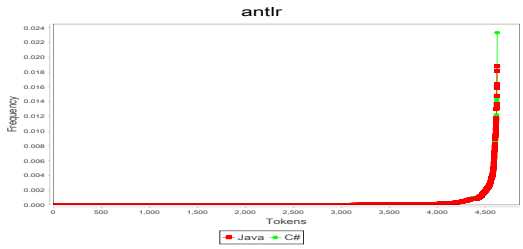


(b) Sorted Token Distribution

Figure 1: Token Distribution (Android vs. iOS)



(a) Token Distribution



(b) Sorted Token Distribution

Figure 2: Token Distribution (Java vs. C#)

3.1 RQ1: Identifier Frequency Distribution

We plotted the token distribution of cross-platform apps. Since all the apps in our experiment appear similarly, we use one of them (*e.g.*, WordPress) to illustrate the problems. Figure 1 shows the token distribution of WordPress's Android and iOS versions. In a similar way, we plotted the token distribution of desktop cross-language apps as a control. Figure 2 shows the token distribution of ANTLR3's Java and C# versions.

Figure 1(a) shows the original token distribution. Its horizontal axis denotes the tokens with the lexicographical order, and its vertical axis denotes the frequency of the token. The distribution indicates that tokens that appear frequently in Android versions often do not appear frequently in iOS versions, since that the programming languages and frameworks of the two platforms are quite different. Figure 2(a) shows the original token distribution of ANTLR3's two language versions. This distribution indicates that some tokens which appear frequently in Java versions also appear frequently in C# ones, as far as desktop projects are concerned. The token distribution distance of WordPress is 1.367 (see Table 4), while the token distribution distance of ANTLR3 is 0.252 (see Table 2). This reconfirms that the difference between the two token distributions of mobile apps is larger than desktop projects.

Figure 1(b) shows the token distribution sorted by the value of frequency. The horizontal axis denotes the tokens that can be different for two versions. It is inspiring that the curves in this figure coincide with each other as those in Figure 2(b). Therefore, we speculate that there may exist a certain mapping between the high-frequency tokens of each version. These mapped tokens may take over the same task in each platform versions. Towards cross-platform clone detection, we plan to mine the mappings based on the distribution in future work.

A similar result is found, when we plotted the token distributions for other apps and desktop projects.

3.2 RQ2: Identifier Similarity

For the nine mobile apps on Android and iOS, their token distribution distances are shown in Table 3. The token distribution distance of Facebook is quite larger than those of the others, while Cordova has the smallest distance.

Furthermore, we calculated the token distribution of the other nine cross-platform apps on Android, iOS and Windows Phone by the pairwise comparison. Table 4 lists the pairwise token distribution distances for the three platforms of the other nine apps. We can see that for the same app, the distance between iOS and Windows Phone is the largest. For six out of nine apps, the distances between Android and Windows Phone are shorter than those between Android and iOS, while two out of nine apps lead to the opposite results. For one out of nine apps, the distances between Android and Windows Phone are almost the same as that between Android and iOS. Through the pairwise comparison, we find that identifiers of apps on Android and on Windows Phone are relatively more similar.

We notice that compared with the distances of the cross-language desktop projects in column "Distance" in Table 2, the distances of cross-platform mobile apps are quite large.

3.3 RQ3: The Feasibility for Clone Detection

As listed in Table 2, ANTLR3 has the smallest distance, while Log4j/Log4net has the largest distance. Figure 3 [3] shows the accumulated clone ratio distribution, *w.r.t.*, the *diff* distance. This is a result of our previous work. The high the clone ratio is, the more accurate the clones detected by *diff* identifier similarity are. For the same *diff* distance, the order of the five projects, whose accumulated clone ratios are from high to low, is ANTLR3, Lucene, FpML, Spring and Log4j/Log4net. The order coincides with the one ordered by distance from low to high. Therefore, the smaller token distribution distance is, the more accurate the clones detected by *diff* identifier similarity are.

For the 18 cross-platform apps, the token distribution distances are relatively larger than the desktop ones. We manually investigate the code of small-size apps (*i.e.*, RioBus, and Adsota), and we find that these apps do not have code that implements the same functionalities. In summary, based on our results, due to the difference of programming languages and frameworks, an identifier-based approach is unlikely to obtain the accurate results for mobile apps.

Table 3: Distance Between Android and iOS Apps

Application	Distance
BlueSenseNetworks	1.514
Cordova	1.063
CycleStreets	1.621
Digipost	1.832
DuckDuckGo	1.757
Facebook	2.662
IRCCloud	1.256
OwnTracks	1.693
RioBus	1.386

Table 4: Pairwise Distance

Application	Distance		
	Andr vs. iOS	iOS vs. WP	WP vs. Andr
Adsota	0.940	1.170	1.469
BlinkID	1.499	1.790	1.285
LowaCodeCamp	1.128	1.231	0.923
OneSignal	1.079	1.299	1.283
OpenPKW	1.738	1.860	1.541
OwnCloud	1.425	1.657	1.439
VK	1.314	1.401	1.059
Wit	1.169	1.182	0.701
WordPress	1.367	1.563	1.170

As we referred in Section 3.1, tokens may not be so similar directly, based on the token frequency in the app, there may exist a certain mapping between the tokens whose frequencies are close to each other. If the identifier similarity is redefined by taking the mapping into consideration, the high accuracy the clones detection can be achieved.

4. RELATED WORK

Kraft *et al.* [11] conduct the first study on cross-language code clones. They implemented a tool called C2D2 based on the CodeDOM library in the Microsoft .NET framework, which uses NRefactory Library to generate the Unified CodeDOM graph for both C# and VB.NET. Al-omari *et al.* [1] also propose a clone detection approach for the .NET language family as well, based on the Common Intermediate Language (CIL). It can detect cross-language clone pairs in C#, J#, and VB.NET. This work focuses on desktop projects and is based on the fact that different programming languages share a common intermediate language. Zhou *et al.* [14] and Chen *et al.* [2] perform Android application clone detections based on static code analysis, but they just focus on one platform.

5. CONCLUSION AND FUTURE WORK

This paper presents an empirical study on identifier similarity in cross-platform mobile apps. Through comparing the identifier similarity, we target at providing insights about whether it is feasible to detect cross-platform clones based on identifier similarity. Our experimental results show that it is unlikely to obtain the accurate results if detecting cross-platform clones based on only identifier similarity. However, if the identifier similarity is redefined by taking into consideration the identifier mapping (*e.g.*, keyword `this` in Java will be mapped to `self` in Objective-C), which are related to frequency, more high accuracy it will achieve. Therefore, in future work, we will mine the identifier mapping based on their frequency, and go towards detecting cross-platform clones. Since there exist many kinds of techniques to detect clones, we also consider function-based or input/output-based clone detection as our future work.

6. ACKNOWLEDGMENTS

This work is sponsored by the 973 Program in China (No. 2015CB352203), the National Nature Science Foundation of China (No. 61572312, No. 61572313, and No. 61272102) and the grant of Science and Technology Commission of Shanghai Municipality (No. 15DZ1100305). This work is performed during Xiao Cheng's visit to Singapore Management University (SMU) and partially supported by SMU.

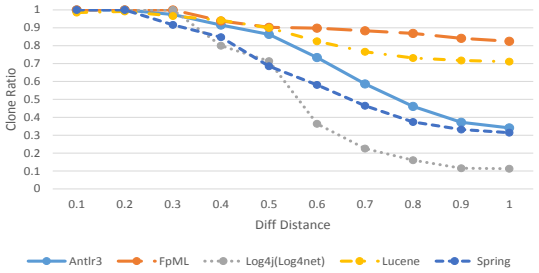


Figure 3: Clone Ratio Distribution

7. REFERENCES

[1] F. Al-Omari, I. Keivanloo, C. K. Roy, and J. Rilling. Detecting clones across microsoft .net programming languages. In *Proc. WCRE*, pages 405–414, 2012.

[2] K. Chen, P. Liu, and Y. Zhang. Achieving accuracy and scalability simultaneously in detecting application clones on android markets. In *Proc. ICSE*, pages 175–186, 2014.

[3] X. Cheng, Z. Peng, L. Jiang, H. Zhong, H. Yu, and J. Zhao. Detecting cross-language clones without intermediates. In *Proc. ASE*, 2016 (to appear).

[4] J. Han, Q. Yan, D. Gao, J. Zhou, and R. H. Deng. Comparing mobile privacy protection through cross-platform applications. In *Proc. NDSS*, 2013.

[5] Z. S. Harris. Distributional structure. *Word*, 10(2-3):146–162, 1954.

[6] L. Jiang, G. Misherghi, Z. Su, and S. Glondu. DECKARD: scalable and accurate tree-based detection of code clones. In *ICSE*, pages 96–105, 2007.

[7] L. Jiang and Z. Su. Automatic mining of functionally equivalent code fragments via random testing. In *Proc. ISSTA*, pages 81–92, 2009.

[8] E. Jürgens, F. Deissenboeck, and B. Hummel. Clonedetective - A workbench for clone detection research. In *Proc. ICSE*, pages 603–606, 2009.

[9] T. Kamiya, S. Kusumoto, and K. Inoue. Ccfinder: A multilinguistic token-based code clone detection system for large scale source code. *IEEE Transaction on Software Engineering*, 28(7):654–670, 2002.

[10] H. Kim, Y. Jung, S. Kim, and K. Yi. MeCC: memory comparison-based clone detector. In *Proc. ICSE*, pages 301–310, 2011.

[11] N. A. Kraft, B. W. Bonds, and R. K. Smith. Cross-language clone detection. In *Proc. SEKE*, pages 54–59, 2008.

[12] S. Kullback and R. A. Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[13] H. Zhong, S. Thummalapenta, T. Xie, L. Zhang, and Q. Wang. Mining API mapping for language migration. In *Proc. 32nd ICSE*, pages 195–204, 2010.

[14] W. Zhou, Y. Zhou, X. Jiang, and P. Ning. Detecting repackaged smartphone applications in third-party android marketplaces. In *Proc. CODASPY*, pages 317–326, 2012.