Prediction and Ranking of Co-change Candidates for Clones

Manishankar Mondal Chanchal K. Roy Kevin A. Schneider
Software Research Laboratory, University of Saskatchewan, Canada
{mshankar.mondal, chanchal.roy, kevin.schneider} @usask.ca

ABSTRACT

Code clones are identical or similar code fragments scattered in a code-base. A group of code fragments that are similar to one another form a clone group. Clones in a particular group often need to be changed together (i.e., co-changed) consistently. However, all clones in a group might not require consistent changes, because some clone fragments might evolve independently. Thus, while changing a particular clone fragment, it is important for a programmer to know which other clone fragments in the same group should be consistently co-changed with that particular clone fragment.

In this research work, we empirically investigate whether we can automatically predict and rank these other clone fragments (i.e., the co-change candidates) from a clone group while making changes to a particular clone fragment in this group. For prediction and ranking we automatically retrieve and infer evolutionary coupling among clones by mining the past clone evolution history. Our experimental result on six subject systems written in two different programming languages (C, and Java) considering both exact and near-miss clones implies that we can automatically predict and rank co-change candidates for clones by analyzing evolutionary coupling. Our ranking mechanism can help programmers pinpoint the likely co-change candidates while changing a particular clone fragment and thus, can help us to better manage software clones.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement

General Terms

Measurement and Experimentation

Keywords

Evolutionary Coupling, Code Clones, Co-change Candidates, Co-change Frequency, Co-change Recency, Ranking

MSR'14, May 31 – June 1, 2014, Hyderabad, India
Copyright 2014 ACM 978-1-4503-2863-0/14/05...$15.00
http://dx.doi.org/10.1145/2597073.2597104

1. INTRODUCTION

Code cloning is a common phenomenon during software development and maintenance. Cloning refers to the process of copying a code fragment from one place of a code-base and pasting it to one or more other places with or without modifications. The original code fragment and the pasted code fragments become clones of one another. A group of code fragments that are exactly or nearly similar to one another form a clone class or a clone group.

Motivation. Code cloning is a matter of great importance from the maintenance perspective. A great many studies [2,3,9,10,12–17,21] have already been done focusing on the impacts of code clones in software development and maintenance. While a number of studies [2,10,12,14,15] indicate some positive effects of cloning during development, there are strong empirical evidences [3,16,17,21] of the negative impacts (such as hidden bug propagation, unintentional inconsistent changes, late propagation, high instability) of clones on software maintenance. The negative impacts indicate the necessity of proper management of code clones for better software maintenance. Clone refactoring is one possible way of managing clones, however, refactoring of all clones in a software system is impractical [13]. There can be situations where refactoring of clones in a particular class is impossible but the clones need to be updated together (i.e., co-changed) consistently [5]. To address this issue, studies have focused on automatic tracking of clone fragments in the clone classes. The idea is that while changing a particular clone fragment in a particular class, a developer will be notified about all other clone fragments in that clone class so that he/she can look at these other clone fragments to decide whether the changes need to be propagated to these fragments too.

However, all clone fragments in a clone class might not need to be updated consistently because clone fragments might evolve independently. Thus, it is important to have a prior knowledge about which other clone fragments in a clone class need to be consistently co-changed while changing a particular clone fragment in that class. Without such prior knowledge a developer can be overwhelmed while dealing with a clone class with a large number of clone fragments (such as 76 clones in a class of our candidate system jEdit) and might need to spend a significant amount of time and effort in understanding and determining which other clone fragments in the clone class need to be consistently co-changed. Focusing on this issue we propose to automatically rank the other clone fragments in a clone class accord-

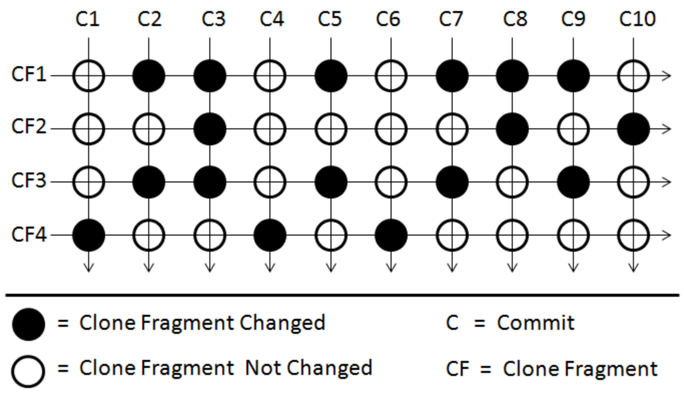


Figure 1: Evolution history of four clone fragments from the same clone class

ing to their probability of being co-changed by mining and inferring from their previous evolution history.

Importance of our study with respect to the existing studies. There is no existing study on such a ranking of the possible co-change candidates for clones. There is an existing tool called *CloneTracker* [6] that presents the other clones in a clone class while a programmer changes a particular clone fragment in that class. However, this tool does not support ranking the clone fragments by their need to co-change consistently. There are also several other clone tracking tools [11, 18, 29]. However, none of these tools supports ranking of co-change candidates for clones. There is no previous study on the possibility of ranking the co-change candidates for clones to assist programmers in dealing with consistent updates of clone fragments. Thus, we believe that our study in this paper is unique and important.

The underlying idea of ranking. Our idea of ranking on the basis of the past evolution history is illustrated in Fig. 1. We can see the evolution history of four clone fragments *CF1*, *CF2*, *CF3*, and *CF4* in a particular clone class *CC* through ten commit operations (*C1* to *C10*). The clone fragments *CF1* and *CF3* changed together (i.e., co-changed) most times. In other words, these two clone fragments have a high tendency of changing together. Thus, it is highly probable that these two clone fragments have co-changed consistently [20]. Also, it is likely that a future change in any one of these two fragments will accompany a corresponding change in the other one. *CF2* has rarely co-changed with *CF1* and *CF3*. *CF4* has never co-changed with any other fragments in its class. In this case, it is likely that *CF4* has a tendency of experiencing independent evolution. After a certain period of evolution *CF4* might not be considered as a clone in the clone class *CC*.

Given this evolution history, if a developer makes a change to the clone fragment *CF1* at a later time, he/she should at first look at *CF3* to check whether *CF3* needs a corresponding change, because *CF3* has co-changed with *CF1* most frequently. CF2 has a comparatively lower probability (compared to CF3) of getting a corresponding change, because *CF2* has co-changed with *CF1* less frequently. Finally, CF4 has the lowest probability. Thus, the co-change candidates of *CF1* can be ordered as: *CF3*, *CF2*, *CF4* according to their tendency of getting co-changed with *CF1*. In this way, as a software system evolves, we can store the past co-change history of the clone fragments and infer this history to make decisions regarding co-change candidates in the future. In this example we present a ranking on the basis of co-change frequency of *CF* with the other clone fragments. However, we also investigate ranking on the basis of co-change recency (i.e., how lately the other clone fragments co-changed with *CF*). We describe this in Section 4.2.

The co-changing tendency of the related program entities (such as files, classes, methods) is known as evolutionary coupling in the literature [8]. We apply the concept of evolutionary coupling in ranking the co-change candidates for clones. Evolutionary coupling is discussed in Section 2.

Findings. We performed our investigation on six subject systems written in two different programming languages (Java, and C) and answer four important research questions listed in Table 1. According to our observation, ranking of co-change candidates for clones is very important because a clone class may contain a large number of clone fragments. The sizes of the largest clone classes of our subject systems Ctags, QMailAdmin, jEdit, Freecol, Carol, and Jabref are respectively 22, 9, 57, 76, 40, and 65. Also, considering all the systems overall 4% of the clone classes contain more than 10 clone fragments. Thus, we believe that automatic ranking of co-change candidates for clones can help programmers identify which other clone fragments from a clone class actually need to be co-changed while changing a particular clone in that class with significantly less effort and time. Even if a clone class contains only a few clones (such as three or four), our ranking mechanism can still help programmers by pin-pointing the most likely co-change candidates for a particular clone fragment being changed. According to our experimental results and analysis we can state that:

While changing a particular clone fragment CF from a particular clone class, we can automatically rank its possible co-change candidates (i.e., the other clone fragments in the same clone class) according to its co-change tendency with them. For ranking we automatically retrieve and infer the evolutionary coupling of CF with its possible co-change candidates from the previous evolution history. We propose a composite ranking mechanism for ranking the possible co-change candidates of CF. Our empirical study shows that our proposed ranking mechanism can assign higher ranks to the actual co-change candidates (i.e., the other clone fragments from the same clone class that actually co-changed with CF) so that a developer attempting to change CF can identify the more likely co-change candidates with less effort and time. The ranking mechanism is our primary contribution. The state of the art techniques and tools [6, 11, 18, 29] do not rank the possible co-change candidates for clones. We believe that our proposed ranking mechanism can complement existing clone tracking tools and techniques.

The rest of the paper is organized as follows. Section 2 describes the terminology, Section 3 discusses the experimental steps, we present and analyze our experimental results in Section 4, Section 5 mentions some possible threats to validity, Section 6 elaborates on the related work, and we conclude our paper by mentioning future work in Section 7.

2. TERMINOLOGY

Types of clones. We conduct our experiment considering exact (Type 1) and near-miss clones (Type 2 and Type 3 clones). As is defined in the literature [23], if two or more clone fragments in a particular clone class are exactly the same disregarding the comments and indentations, these

Table 1: Research Questions

SL	Research Question
1	Can we predict co-change candidates for a particular clone fragment by using evolutionary coupling?
2	Can we achieve better ranking of co-change candidates that exhibited evolutionary coupling by considering co-change recency instead of co-change frequency?
3	What are the characteristics of the clone fragments that exhibit evolutionary coupling? Which characteristic can help us in better ranking of co-change candidates that have not yet exhibited evolutionary coupling?
4	How can we rank both types of co-change candidates - (1) the candidates that exhibited evolutionary coupling, and (2) the candidates that did not exhibit evolutionary coupling for a particular clone fragment?

Table 2: Subject Systems

Sys.	Lang.	Domains	LOC	Revs
Ctags	C	Code Def. Generator	33,270	774
QMail Admin	C	Mail Management	4,054	317
jEdit	Java	Text Editor	191,804	4000
Freecol	Java	Game	91,626	1950
Carol	Java	Game	25,091	1700
Jabref	Java	Reference Manager	45,515	1545

Revs = Revisions. Sys = Systems

clone fragments are called exact clones (i.e., Type 1 clones) of one another. Type 2 clones are syntactically similar code fragments. In general, Type 2 clones are created from Type 1 clones because of renaming variables or changing data types. Type 3 clones are mainly created because of additions, deletions, or modifications of lines in Type 1 or Type 2 clones.

Evolutionary coupling. During the evolution of a software system if two or more program entities (such as files, classes, methods) appear to change together (i.e., co-change) frequently (i.e., in many commits) then we say that these entities exhibit evolutionary coupling. It is likely that these entities are related and a future change to any one of these entities will accompany corresponding changes to the other entities. By mining and analyzing evolutionary coupling we can discover the underlying relationships among program entities in a software system [8, 31]. Evolutionary coupling helps us predict the co-change candidates (i.e., the other entities that might also need to be changed) while changing a particular entity [31]. In this research work, we apply the concept of evolutionary coupling to discover the underlying relationships among clones and to predict the co-change candidates while changing a particular clone fragment in a particular clone class.

In order to mine evolutionary coupling among clone fragments we determine all possible pairs of co-changed clone fragments by examining the software evolution history.

Pair of Co-changed Clone Fragments (PCCF). A pair of co-changed clone fragments (i.e., a *PCCF*) consists of two clone fragments *CF*1 and *CF*2 from the same clone class such that they changed together (i.e., co-changed) in at least one commit operation during system evolution.

During the evolution of a software system if $n >= 2$ clone fragments from a particular clone class changed together (co-changed) in a particular commit, we determine all possible pairs from these n clone fragments. Each of these pairs is a *PCCF*. For every *PCCF* that we obtain by mining the whole evolution history, we determine the number of times (i.e., the number of commits) the constituent clones co-changed. We call this number the co-change frequency of a *PCCF*.

3. EXPERIMENTAL STEPS

In this experiment, we consider clones residing in methods. Thus, both fully cloned and partially cloned methods have been investigated. For a particular subject system, we collect all of its revisions (mentioned in Table 2), then extract the methods in each revision using CTAGS, and then determine method genealogies following the technique proposed by Lozano and Wermelinger [16]. We detect clones in each revision using NiCad [4] and then map the clones to the already detected methods of the corresponding revisions. As method genealogies are already detected, after clone mapping we can easily track the evolution of each clone fragment residing inside a method. Finally, we detect changes between every two consecutive revisions and map these changes to the methods as well as clones located inside the methods. For the details of these steps we refer the readers to our earlier work [19]. We detected both exact and near-miss block clones using the NiCad clone detector considering a dissimilarity threshold of 30% with blind renaming. This setting of NiCad is considered standard for detecting near-miss clones [24].

After the preliminary steps we determine all possible pairs of co-changed clone fragments (*PCCF*s) by examining all the commit operations. For each of the *PCCF*s we determine its co-change frequency. We rank the possible co-change candidates of a particular clone fragment on the basis of this co-change frequency. However, we also rank the possible co-change candidates on the basis of co-change recency (i.e., on the basis of how lately a co-change occurred). We describe and compare these two ranking mechanisms in Section 4.2.

4. EXPERIMENTAL RESULTS AND ANALYSIS

We perform our investigation on each of the subject systems listed in Table 2. We determine all pairs of co-changed clone fragments (*PCCF*s) from these candidate systems. For each of the systems we determine four measures - (1) total number of clones created during system evolution, (2) total number of clones that received changes (at least once) during evolution, (3) total number of clones that exhibited evolutionary coupling, and (4) total number of *PCCF*s and show these in Table 3. If a clone fragment is included in at least one *PCCF*, we consider that it has exhibited evolutionary coupling, because it has co-changed with at least one of the other clone fragments in its clone class during evolution.

We also determine the percentage of modified clones (i.e., the clones that received changes at least once during evolution) that exhibited evolutionary coupling. This percentage for each subject system is shown in Fig. 2. The figure shows that in case of each of the subject systems, a consid-

Table 3: Statistics Regarding Evolutionary Coupling among Clones

Systems	NC	NCRC	NCEC	NPCCF
Ctags	694	412	89	79
QMailAdmin	137	128	34	347
jEdit	12329	1356	326	324
Freecol	2265	1659	489	577
Carol	3040	1365	616	2041
Jabref	3708	1552	509	684

NC = No. of Clones created during system evolution.
NCRC = No. of Clones that Received Changes.
NCEC = No. of Clones that showed Evolutionary Coupling.
NPCCF = No. of the pairs of co-changed clone fragments.

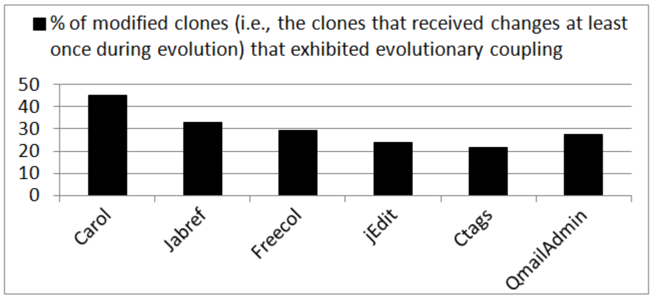


Figure 2: Proportion of modified clones (i.e., the clones that received changes at least once during evolution) that exhibited evolutionary coupling

erable percentage of modified clones exhibited evolutionary coupling. The overall percentage considering all subject systems is 31.85%. Thus, it seems that overall 68.15% of the modified clones (i.e., clone fragments that received changes) evolved independently. However, according to our observation, overall 70.82% of the total clone fragments of a subject system never changed during evolution.

We automatically retrieve evolutionary coupling from each of the candidate subject systems. The XML files containing the pairs of co-changed clone fragments are available online[1]. Our primary goal in this research work is to investigate whether we can predict and rank co-change candidates for clones using evolutionary coupling. In the following subsections, we answer four research questions regarding this.

4.1 Answering Research Question 1

RQ 1: Can we predict co-change candidates for a particular clone fragment using evolutionary coupling?

Finding the answer to this research question is the central objective of our research work. Here, we should note that by the term 'co-change candidates' for a particular clone fragment we mean the other clone fragments in the same clone class containing the particular clone fragment. However, non-clone fragments can also co-change with a clone fragment. We do not investigate this in this research work. We mainly focus on the efficient tracking as well as proper management of code clones. We target to minimize the drawbacks of the existing clone tracking techniques and tools. We plan to investigate regarding the non-clone co-change

[1]XML Files: `https://homepage.usask.ca/~mam815/ongoingresearch.php`

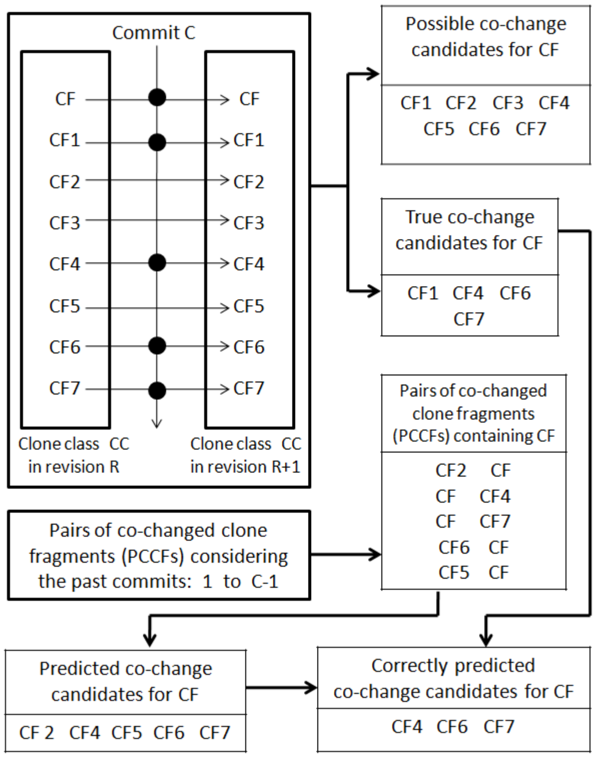

Commit C
CF
CF1
CF2
CF3
CF4
CF5
CF6
CF7
CF
CF1
CF2
CF3
CF4
CF5
CF6
CF7
Clone class CC in revision R
Clone class CC in revision R+1
Possible co-change candidates for CF
CF1 CF2 CF3 CF4 CF5 CF6 CF7
True co-change candidates for CF
CF1 CF4 CF6 CF7
Pairs of co-changed clone fragments (PCCFs) containing CF
CF2 CF
CF CF4
CF CF7
CF6 CF
CF5 CF
Pairs of co-changed clone fragments (PCCFs) considering the past commits: 1 to C-1
Predicted co-change candidates for CF
CF 2 CF4 CF5 CF6 CF7
Correctly predicted co-change candidates for CF
CF4 CF6 CF7

Figure 3: Prediction of co-change candidates using evolutionary coupling

fragments as future work. In the following paragraphs we describe our investigation methodology for answering **RQ 1**.

Methodology. For answering this research question we automatically analyze each of the commits of a subject system. Let us consider a particular clone class *CC* in a particular revision *R* of a subject system. A commit operation *C* was applied on revision *R* and more than one clone fragments of the clone class *CC* co-changed (i.e., changed together) in this commit. We consider a particular clone fragment *CF* from *CC* such that *CF* changed in commit *C*. So, we know which other clone fragments from clone class *CC* actually co-changed with *CF* in commit *C*. However, we want to determine whether and to what extent we can predict these true co-change candidates for *CF* (in commit *C*) by analyzing the evolutionary coupling exhibited by *CF* during the previous commits 1 to $C-1$.

Our prediction mechanism is presented in Fig. 3. In this figure, we see a clone class *CC* in revision *R*. The clone class *CC* contains eight clone fragments *CF* to *CF7*. A commit operation *C* applied on revision *R* modified five clone fragments - *CF*, *CF1*, *CF4*, *CF6*, and *CF7* from this class. The corresponding clone class in revision $R+1$ (i.e., after the application of the commit operation *C*) is also shown in the figure. We consider the clone fragment *CF*. From the figure we can determine the actual co-change candidates (*CF1*, *CF4*, *CF6*, and *CF7*) for *CF*. We want to determine the extent we can correctly predict these actual co-change candidates for *CF* by analyzing its evolutionary coupling during the past commits (i.e., the commits from 1 to $C-1$). For the purpose of describing we define the following four sets considering the clone fragment *CF* and the commit operation *C*.

Possible Co-change Candidates. All the clone fragments of the clone class *CC* excluding *CF* are termed as the set of *possible co-change candidates* of *CF*. Each of these clone fragments has a possibility of co-changing with *CF*. However, all of these possible co-change candidates might not co-change with *CF* in a particular commit.

True Co-change Candidates. All those clone fragments (from the clone class *CC*) that actually co-changed with *CF* in commit operation *C* are termed as the set of *true co-change candidates* of *CF* in commit *C*.

Predicted Co-change Candidates. All those clone fragments (in the clone class *CC*) that we can predict as the co-change candidates for *CF* by analyzing the evolutionary coupling of *CF* in previous commits (from commit 1 to $C-1$) are termed as the set of *predicted co-change candidates*.

Correctly Predicted Co-change Candidates. All those clone fragments from the set of *predicted co-change candidates* that actually co-changed with *CF* in commit *C* are termed as the set of *correctly predicted co-change candidates*.

Fig. 3 shows these four sets for clone fragment *CF* considering commit *C*. We determine the set of *predicted co-change candidates* for *CF* in the following way. We at first retrieve all the pairs of co-changed clone fragments (*PCCF*s) considering all the commits from 1 to $C-1$. We select those *PCCF*s where the clone fragment *CF* appears. Fig. 3 shows five such *PCCF*s. From these *PCCF*s we determine all other clone fragments beside *CF*. These clone fragments (*CF2*, *CF4*, *CF5*, *CF6* and *CF7* as shown in Fig. 3) are the set of *predicted co-change candidates* for *CF*, because we get these by analyzing the evolutionary coupling of *CF*.

Finally, we determine the set of *correctly predicted co-change candidates* for *CF* in commit *C*. Fig. 3 shows three correctly predicted co-change candidates - *CF4, CF6, CF7*. We determine the following six measures for *CF* considering the commit operation *C*.

(1) The number of possible co-change candidates for *CF*,

(2) The number of possible co-change candidates that exhibited evolutionary coupling with *CF* in the previous commits 1 to $C-1$. This is the number of predicted co-change candidates for *CF* in commit *C*

(3) The number of possible co-change candidates that did not exhibit evolutionary coupling with *CF* in the past commits. We call these co-change candidates the non-predicted co-change candidates for *CF* in commit *C*, because we could not predict them by analyzing evolutionary coupling.

(4) The number of predicted co-change candidates that actually co-changed with *CF* in commit *C*. This is the number of correctly predicted co-change candidates for *CF*.

(5) The number of non-predicted co-change candidates that co-changed with *CF* in commit *C*.

(6) The number of true co-change candidates (i.e., the possible co-change candidates that actually co-changed with *CF*) for *CF* in commit *C*.

We examine all the commit operations where more than one clone fragments from the same clone class changed together (i.e., co-changed). For each of the clone fragments (*CF*) that changed in such a commit, we determine the above six measures. Considering all the commit operations of a particular subject system we determine the summation for each of the respective measures. Then, we calculate the following percentages using these measures.

(1) The percentage of possible co-change candidates that were selected as the predicted co-change candidates. In other words, the percentage of possible co-change candidates that exhibited evolutionary coupling with *CF*.

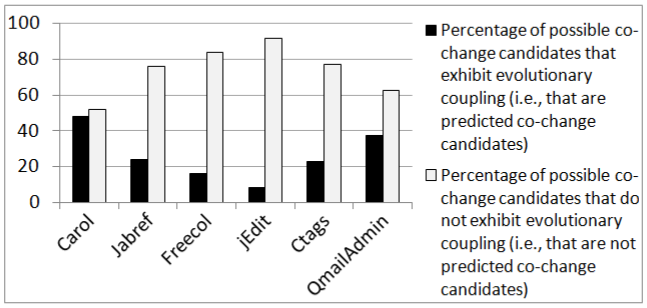


Figure 4: Comparison between the proportions of predicted co-change candidates and non-predicted co-change candidates

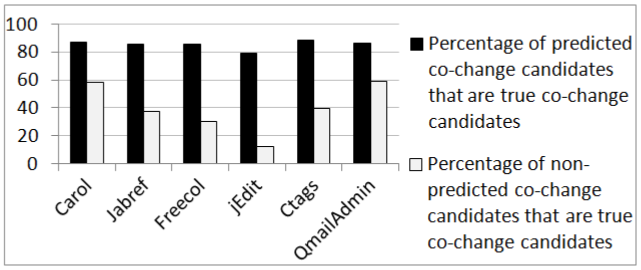


Figure 5: Comparison between the percentage of predicted co-change candidates that were true co-change candidates (i.e., the precision) and the percentage of non-predicted co-change candidates that were true co-change candidates

(2) The percentage of possible co-change candidates that were selected as the non-predicted co-change candidates. In other words, the percentage of possible co-change candidates that did not exhibit evolutionary coupling with *CF*.

(3) The percentage of predicted co-change candidates that are true co-change candidates. We also call this the *precision in predicting true co-change candidates by analyzing evolutionary coupling.*

(4) The percentage of non-predicted co-change candidates that are true co-change candidates.

(5) The percentage of true co-change candidates that we could predict (i.e., by analyzing evolutionary coupling). We also call this percentage the *recall in predicting true co-change candidates by analyzing evolutionary coupling.*

Fig. 4 compares the first two percentages - (1) the percentage of possible co-change candidates that exhibited evolutionary coupling, and (2) the percentage of possible co-change candidates that did not exhibit evolutionary coupling, for each of the subject systems. We see that the proportion of possible co-change candidates that exhibited evolutionary coupling is much lower than its counter part for most of the subject systems. The overall values of these percentages are, 26% and 74% respectively.

However, from the comparison scenario in Fig. 5 (comparing the third and fourth percentages) we see that the proportion of predicted co-change candidates that were true co-change candidates (i.e., that were correctly predicted) is much higher compared to the proportion of non-predicted co-change candidates that were selected as true co-change candidates for most of the subject systems. *Thus, the predicted co-change candidates have a much higher probability of being true co-change candidates. In other words, the pos-*

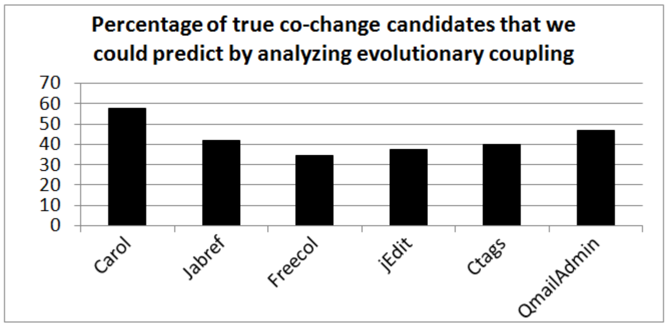


Figure 6: The proportion of true co-change candidates that we could predict by analyzing evolutionary coupling (i.e., the recall)

sible co-change candidates that exhibited evolutionary coupling have a much higher probability of being true co-change candidates compared to the possible co-change candidates that did not exhibit evolutionary coupling.

Finally, Fig. 6 demonstrates that *for each of the subject systems a considerable proportion of true co-change candidates can be predicted by analyzing evolutionary coupling.* The overall proportion considering all subject systems is 43.17%. As we mentioned before, this percentage is the recall in predicting true co-change candidates by analyzing evolutionary coupling. We also determine the overall precision (i.e., the third percentage) in predicting true co-change candidates considering all systems. This overall precision is 85.18%.

Answer to RQ 1. From our analysis and discussion we can say that evolutionary coupling can help us predict true co-change candidates for a particular clone fragment with considerable accuracy in terms of precision (= 85.18%) and recall (= 43.17%).

4.2 Answering Research Question 2

RQ 2: Can we achieve better ranking of co-change candidates by considering co-change recency instead of co-change frequency?

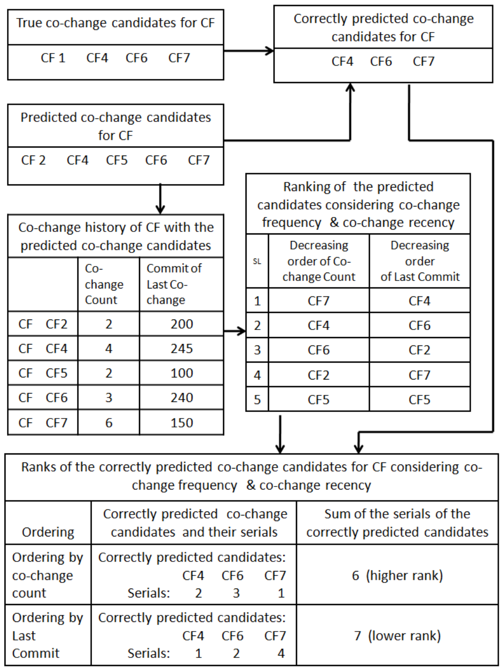
From the first research question we understand that we can predict a considerable amount of true co-change candidates for clones by analyzing evolution coupling. For answering this research question we rank the predicted co-change candidates (i.e., predicted by analyzing evolutionary coupling) in the following two ways.

(1) On the basis of co-change frequency

(2) On the basis of co-change recency (i.e., how lately a co-change occurred)

Then, we determine which ranking system generally gives better ranks for the correctly predicted co-change candidates. In the following paragraphs we at first describe the ranking mechanisms and then discuss their comparison.

Description of the ranking mechanisms. We at first assume a particular clone class *CC* in a particular revision *R* of a particular candidate system. More than one clone fragments from this class co-changed in the commit operation *C* applied on revision *R*. We consider a particular clone fragment *CF* from *CC* that changed in this commit operation *C*. For the clone fragment CF, we determine the following two sets of co-change candidates considering commit operation C following the methodology described in the previous subsection.



True co-change candidates for CF
CF 1 CF4 CF6 CF7

Correctly predicted co-change candidates for CF
CF4 CF6 CF7

Predicted co-change candidates for CF
CF 2 CF4 CF5 CF6 CF7

Co-change history of CF with the predicted co-change candidates		
	Co-change Count	Commit of Last Co-change
CF CF2	2	200
CF CF4	4	245
CF CF5	2	100
CF CF6	3	240
CF CF7	6	150

Ranking of the predicted candidates considering co-change frequency & co-change recency		
SL	Decreasing order of Co-change Count	Decreasing order of Last Commit
1	CF7	CF4
2	CF4	CF6
3	CF6	CF2
4	CF2	CF7
5	CF5	CF5

Ranks of the correctly predicted co-change candidates for CF considering co-change frequency & co-change recency		
Ordering	Correctly predicted co-change candidates and their serials	Sum of the serials of the correctly predicted candidates
Ordering by co-change count	Correctly predicted candidates: CF4 CF6 CF7 Serials: 2 3 1	6 (higher rank)
Ordering by Last Commit	Correctly predicted candidates: CF4 CF6 CF7 Serials: 1 2 4	7 (lower rank)

Figure 7: Ranking of predicted co-change candidates by co-change frequency and co-change recency

(1) **True co-change candidates.** The clone fragments from clone class *CC* that actually co-changed with *CF*

(2) **Predicted co-change candidates.** The clones in clone class *CC* that exhibited evolutionary coupling with *CF*

We rank these predicted co-change candidates in two ways mentioned above. The ranking procedures are as follows.

Ranking predicted co-change candidates on the basis of co-change frequency. We know that each of the predicted co-change candidates has previously (in the commits preceding the commit *C*) co-changed with the clone fragment *CF*. We assume higher ranks for those predicted co-change candidates that previously co-changed with *CF* more frequently. We sort these predicted co-change candidates in decreasing order of their co-change counts with *CF*.

Ranking predicted co-change candidates on the basis of co-change recency. In this case the predicted co-change candidates that co-changed with *CF* more recently are given higher ranks. For each of the predicted co-change candidates, we determine the last commit operation where it co-changed with *CF*. We sort the predicted co-change candidates in decreasing order of their last commits.

An example describing the two ways of ranking. Fig. 7 shows an example of the two ways of ranking of the predicted co-change candidates for a clone fragment *CF*. The sets - **(1)** true co-change candidates, **(2)** predicted co-change candidates, and **(3)** correctly predicted co-change candidates for *CF* are taken from Fig. 3. In this figure (i.e., Fig. 7) we show a possible co-change history of *CF* with each of the predicted co-change candidates. The co-change history consists of two pieces of information - **(1)** The number of times *CF* co-changed with a predicted co-change candidate, and **(2)** The last commit operation where *CF* co-changed with a predicted co-change candidate.

These are the indicators of co-change frequency and co-change recency respectively. We automatically collect this information from the co-change history of *CF*. From the figure (Fig. 7) we see that *CF* co-changed with *CF7* the highest number of times (i.e., 6 as shown in the figure). However, among all the predicted co-change candidates, *CF4* co-changed with *CF* most recently. This co-change occurred in commit 245 as shown in the figure. Rankings (i.e., orderings) of the predicted co-change candidates in two ways (i.e., on the basis of co-change frequency and co-change recency) are also shown in Fig. 7. We see that while *CF7* is assigned the highest rank (i.e., its serial number is 1) on the basis of co-change frequency, *CF4* is assigned the highest rank on the basis of co-change recency. In both cases, *CF5* gets the lowest rank.

Comparison of the ranking mechanisms. We see that each of the two ranking systems described above is based on evolutionary coupling of the clone fragments. However, we want to determine which one is better. The ranking system that provides better ranks for the correctly predicted co-change candidates should be considered as the superior one.

For each clone fragment *CF* changed in a commit operation C we determine its predicted co-changed candidates and rank these in two ways to get two different rankings (or orderings) of the predicted co-change candidates. Then we determine the correctly predicted co-change candidates and locate them in each of the rankings of the predicted co-change candidates. We determine the serial numbers (i.e., position values) of the correctly predicted co-change candidates from each ranking. We get two sets of serial numbers from the two ranking systems. We determine the summation of the serial numbers obtained from each ranking system. A lower summation indicates better ranking.

In Fig. 7, the widest table (i.e., the bottom one) shows the comparison of the ranking systems considering the predicted co-change candidates for the clone fragment *CF*. There are five predicted co-change candidates for *CF*. According to the example, three (*CF4, CF6, CF7*) of these predicted co-change candidates have actually co-changed with *CF* in commit operation C. We show the serial numbers of the predicted co-change candidates in each of the rankings. We see that from the ranking on the basis of co-change frequency, we get the serial numbers - 2, 3, and 1 for the correctly predicted co-change candidates *CF4*, *CF6*, and *CF7* respectively. The serial numbers obtained from the other ranking system are 1, 2, and 4 respectively. The summations (6 considering co-change frequency, and 7 considering co-change recency) of the serial numbers are also shown in the figure. According to our explanation, the ranking system on the basis of co-change frequency provides better ranks to the correctly predicted co-change candidates.

For each of the clone fragments changed in each of the commits we determine which ranking system provides better ranks to the correctly predicted co-change candidates. We determine two percentages - **(1)** The percentage of cases where the ranking on the basis of co-change frequency gives us better ranks for the correctly predicted co-change candidates, and **(2)** The percentage of cases where the ranking on the basis of co-change recency provides us better ranks for the correctly predicted co-change candidates

We show these percentages in Fig. 8. In case of each of the subject systems, for most of the cases the two ranking systems provide the same ranks to the correctly predicted co-change candidates. However, if we consider the remain-

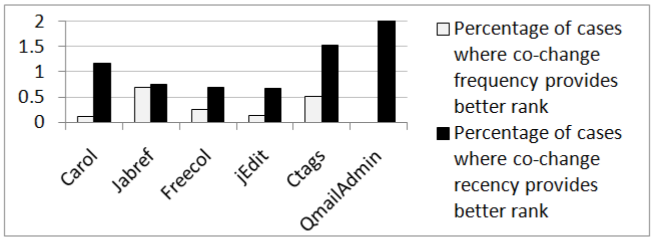


Figure 8: Comparison of the two ranking systems (ranking using co-change frequency and co-change recency)

ing cases we see (c.f., Fig. 8) that *for each of the subject systems, co-change recency provides better ranks to the correctly predicted co-change candidates compared to co-change frequency.*

Answer to RQ 2. From our discussion we decide that we can achieve better ranking of co-change candidates by considering co-change recency instead of co-change frequency.

4.3 Answering Research Question 3

RQ 3: What are the characteristics of the clone fragments that exhibit evolutionary coupling? Which characteristic can help us in better ranking of the co-change candidates that have not yet exhibited evolutionary coupling?

Answering this research question is important. From our discussion and analysis while answering *RQ 1* we understand that the percentage of predicted co-change candidates for a particular clone fragment *CF* (i.e., the percentage of possible co-change candidates that exhibit evolutionary coupling with *CF*) is always smaller compared to the percentage of non-predicted co-change candidates (i.e., the possible co-change candidates that have not yet exhibited evolutionary coupling with *CF*). We can rank the predicted co-change candidates. However, we are also interested in ranking the non-predicted co-change candidates. For this purpose we analyze the characteristics of the clone fragments that exhibit evolutionary coupling. Let us assume that we have discovered a dominant characteristic of the clone fragments that exhibit evolutionary coupling. If we see that some of the non-predicted co-change candidates also possess this characteristic, we can assume better ranks for these non-predicted co-change candidates.

Methodology. For answering this research question, we analyze the following two characteristics of the clone fragments that exhibit evolutionary coupling.

(1) **Regarding clone type.** We analyze whether the clone fragments exhibiting evolutionary coupling are method clones or block clones.

(2) **Regarding the proximity of the clone fragments.** We analyze whether two clone fragments exhibiting evolutionary coupling generally remain in close proximity to each other or not.

Considering each of the subject systems we determine all the pairs of co-changed clone fragments (*PCCF*s). For each *PCCF* we determine the types (method clone or block clone) of the two participating clone fragments. We also determine whether the two clone fragments remain in the same file or in different files. For each of the subject systems we determine the following two percentages - **(1)** The percentage of *PCCF*s (i.e., the pairs of co-changed clone fragments) where the participating clone fragments remain in the same file. **(2)** The percentage of *PCCF*s where both of the participating clone fragments are method clones.

We present these two percentages in Fig. 9. From the black bars we see that for most of the subject systems (except QMailAdmin), the participating clone fragments in most of the *PCCF*s (i.e., above 55% of the *PCCF*s) remain in the same file. We also observe (from the white bars) that in case of four subject systems (Carol, Freecol, jEdit, and Ctags), both of the participating clone fragments in most of the *PCCF*s are method clones (i.e., clones are full methods). However, if we compare these two characteristics, we see that file proximity is the more dominant one for most of the subject systems (except Carol).

We rank the non-predicted co-change candidates (for a particular clone fragment *CF* in a particular commit *C*) in the following two ways.

(1) Considering clone file proximity and

(2) Considering clone type (method clones or block clones)

In case of ranking considering clone file proximity, we provide higher (i.e., better) ranks to those non-predicted co-change candidates that are nearer (i.e., in closer proximity) to the clone fragment *CF*. The clone file proximity between *CF* and a particular non-predicted co-change candidate is determined by the distance between the corresponding container files in the file system structure as was done by Alali et. al [1]. In case of ranking considering clone type, we provide higher ranks to those non-predicted co-change candidates that are method clones. We compared these two ranking systems following the same way described while answering RQ 2. In this case, from the ranking (of the non-predicted co-change candidates) obtained from each ranking system, we determine the summation of the position values (i.e., serial numbers) of the non-predicted true co-change candidates. The ranking system that provides the lower summation (i.e., the higher rank) is the better one.

Considering each of the clone fragments changed in each of the commits (where more than one clone fragments from the same clone class co-changed) of a particular subject system we determine and rank the non-predicted co-change candidates using the above two ranking systems and determine which ranking system provides better ranks to the non-predicted true co-change candidates. We determine two percentages - **(1)** The percentage of cases where the ranking system on the basis of clone file proximity provides better ranks, and **(2)** The percentage of cases where the ranking system on the basis of clone type provides better ranks

These two percentages for each of the subject systems is shown in Fig. 10. We see that in case of each of the subject systems, *the percentage of cases where ranking by clone file proximity provides better ranks is much higher than the percentage of cases where ranking by clone type provides better ranks to the non-predicted true co-change candidates.* However, we observed that for 56% to 87% cases (considering all the subject systems) the two ranking systems provided the same ranks. Finally, we consider the ranking system on the basis of clone file proximity to be the better one for ranking the non-predicted co-change candidates.

Answer to RQ 3. Generally, the clone fragments that exhibit evolutionary coupling - (1) remain in close proximity (i.e., in the same file) to each other, and (2) are method clones. According to our analysis, consideration of clone file proximity can help us in better ranking of the non-predicted co-change candidates for a particular clone fragment.

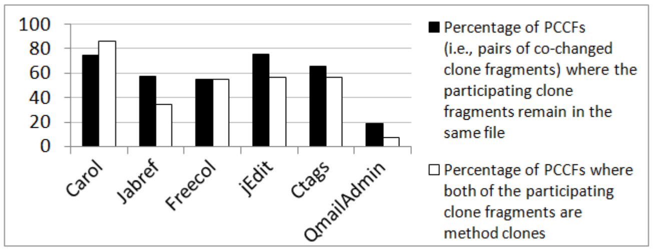


Figure 9: Comparison of the characteristics of the clone fragments that exhibit evolutionary coupling

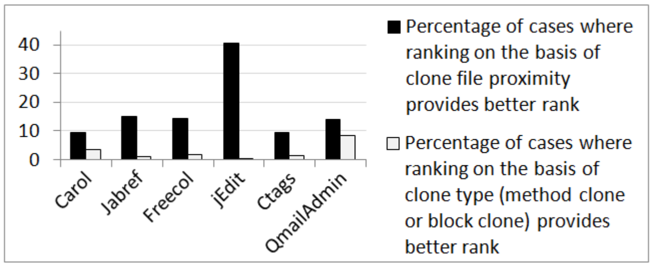


Figure 10: Comparison of the two ranking systems for the non-predicted co-change candidates

4.4 Answering Research Question 4

RQ 4: How can we rank both type of co-change candidates - (1) the candidates that exhibited evolutionary coupling, and (2) the candidates that did not exhibit evolutionary coupling for a particular clone fragment?

Answering this research question is important. From the answers to the previous research questions we understand that the possible co-change candidates for a particular clone fragment can broadly be divided into two disjoint sets - (1) the predicted co-change candidates (i.e., the candidates that exhibited evolutionary coupling with the particular clone fragment), and (2) non-predicted co-change candidates (i.e., the candidates that have not yet exhibited evolutionary coupling with the particular clone fragment). For the first set we decide a ranking mechanism on the basis of co-change recency as the better one. For the second we found the ranking mechanism on the basis of clone file proximity to be the better one. Thus intuitively, a combination of the two ranking mechanisms (ranking by co-change recency for the predicted candidates and ranking by clone file proximity for the non-predicted candidates) can possibly be used for better ranking of all co-change candidates of a particular clone fragment. We verify this in this research question in the following way.

Methodology. We rank all the possible co-change candidates for a particular clone fragment *CF* changed in a particular commit *C* in the following two ways.

(1) Ranking of all possible co-change candidates using a combination of the two ranking mechanisms - ranking of the predicted candidates by co-change recency, and ranking of the non-predicted candidates by clone file proximity

(2) Ranking of all possible co-change candidates by clone file proximity

We compared these two ways of ranking to determine which one can provide better ranks to all the true co-change candidates for a particular clone fragment *CF* in a particular commit *C*. We performed our comparison in the same way (i.e., considering all commit operations) as is described

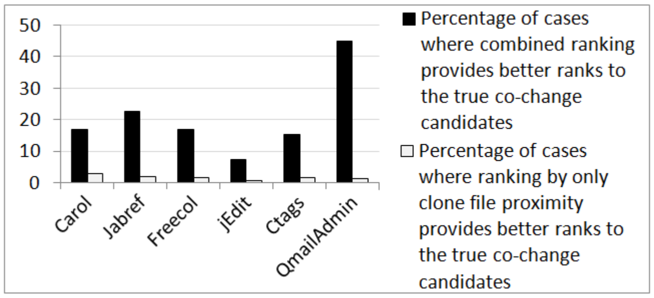


Figure 11: Comparison between the combined ranking (i.e, ranking predicted candidates by co-change recency and non-predicted co-change candidates by clone file proximity) and ranking considering only clone file proximity

while answering RQ 2. We determine the percentage of cases where we get better ranks for the true co-change candidates using the combined ranking mechanism (the first way mentioned above) and also the percentages of cases where the second way of ranking (ranking all candidates by clone file proximity) provides better ranks. These percentages are shown in Fig. 11. From Fig. 11 we see that in case of each of the subject systems, the percentage of cases where the combined ranking provides better ranks is much higher compared to the percentage of cases where ranking by only clone file proximity provides better ranks to the true co-change candidates. Thus, we decide that the combined ranking mechanism yields in better ranking of the true co-change candidates compared to the file proximity ranking.

Answer to RQ 4. From our discussion and analysis presented above, we suggest a combined ranking (ranking of predicted co-change candidates by co-change recency, and ranking of non-predicted co-change candidates by clone file proximity) of the possible co-change candidates for a particular clone fragment.

5. THREATS TO VALIDITY

We used the NiCad clone detector [4] for detecting clones. For different settings of NiCad, the statistics that we present in this paper might be different. Wang et. al [30] defined this problem as the *confounding configuration choice problem* and conducted an empirical study to ameliorate the effects of the problem. However, the settings that we have used for NiCad are considered standard [24] and with these settings NiCad can detect clones with high precision and recall [25, 26].

For determining the prediction accuracy of our implemented prediction system, we have used the precision and recall measures. Shepperd and MacDonell [27] conducted a fine grained study on the evaluation of the prediction systems. According to their observation different prediction systems might give us conflicting results. They advised researchers not to use biased accuracy measures for the purpose of determining prediction accuracy. However, precision and recall are well known and extensively used measures and they represent the exact scenario (in term of accuracy) when reported together. Thus, we believe that our reported findings in the form of precision and recall are important.

The subject systems that we have studied in this experiment are not enough to take a concrete decision regarding the ranking of co-change candidates for clones. However, our candidate systems are of diverse variety in terms of application domains, sizes and revisions. Thus, our findings cannot be attributed to a chance. Finally, we believe that our findings are important and can help us in better management of code clones.

6. RELATED WORK

A great many studies have already been done regarding the detection, evolution [2, 13, 28], impact analysis [3, 10, 12, 14–17, 21, 22], and maintenance [7, 11, 18, 20, 29] of code clones. Although there are some positive impacts [10, 12, 14, 15] of cloning on both software development and maintenance, a number of studies [3, 16, 17, 21] have shown empirical evidences of some strong negative impacts of clones on software evolution. Focusing on the negative impacts, software researchers have emphasized on proper maintenance of code clones through clone refactoring [20] or tracking [6, 7, 11, 18, 29]. As clone refactoring is not always possible [13], tracking of clones becomes very important for better software maintenance. As our research work is focused on clone tracking, we discuss the existing clone tracking techniques and studies.

The most recent study on clone tracking was conducted by Duala-Ekoko and Robillard [7]. They introduced the concept of *clone region descriptor*. On the basis of this concept they proposed a technique for tracking clones in evolving software. They implemented a tool called 'CloneTracker' [6] as an Eclipse plug-in for tracking clones. The tool provides supports for two tasks - (1) change notifications and (2) simultaneous editing of clones. After modifying a particular clone tracked by CloneTracker, the programmer is notified about the other clone fragments in the same group that contains the modified clone. However, 'CloneTracker' does not support any type of ranking of these other clone fragments. In other words, this tool does not have any prior knowledge about which other clone fragments have higher probability of co-changing with the particular clone fragment. Our research work in this paper focuses on ranking of these other clone fragments so that the responsible programmer can easily pinpoint those other clone fragments that are more likely to consistently co-change with the particular clone fragment.

Jablonski and Hou *jablonski* developed a tool called *CReN* to track copy-paste code clones and support consistent renaming of identifiers. Miller and Myer [18] proposed a technique for simultaneous editing in multiple clone fragments in the same clone class to minimize the task of repetitive editing. They implemented their technique in a text editor called *LAPIS*. There is also another clone tracking tool called *Codelink* developed by Toomin et. al [29]. However, none of these existing clone trackers supports ranking of the co-change candidates for clones.

There is an existing study conducted by Mondal et. al [20] regarding the detection and ranking of *SPCP clones* (i.e., clones that evolve following a similarity preserving change pattern) through analysis of evolutionary coupling. *SPCP clones* are important candidates for refactoring. They ranked these *SPCP clones* using evolutionary coupling to prioritize refactoring tasks. However, as clones are not always refactorable, it is important to track them efficiently. Our prime focus in this research work is to support the existing clone trackers by providing facilities for automatic prediction and ranking of the likely co-change candidates when we change a particular clone fragment.

From our discussion above we believe that our study presented in this paper is important and unique. Our experimental result has the potential to assist in better management of code clones and thus can help us in better software maintenance.

7. CONCLUSION

In this research work we present an in-depth investigate on the possibility of predicting and ranking co-change candidates for clones through analysis of evolutionary coupling among clone fragments. We empirically studied six subject systems written in two programming languages (Java and C). We used the NiCad clone detector for detecting clones. Our experimental results and analysis imply that while changing a particular clone fragment in a particular clone class

(1) We can predict which other clone fragments in the same clone class will also co-change with the particular clone fragment with considerable accuracy (precision = 85.18%, recall = 43.17%) by analyzing the evolutionary coupling of the particular clone fragment.

(2) We can automatically rank the possible co-change candidates (all other clone fragments in the same clone class) on the basis of their evolutionary coupling with the particular clone fragment such that the possible co-change candidates that are more likely to co-change with the particular clone fragment get higher ranks.

We propose a composite ranking mechanism for ranking the possible co-change candidates of a particular clone fragment. In the presence of such a ranking, the responsible programmer can easily pinpoint the likely co-change candidates while changing a particular clone. Thus, our ranking mechanism can complement existing clone tracking techniques and tools for better management of code clones. As future research work we plan to implement a clone tracking tool with the capability of automatically ranking possible co-change candidates when changing a particular clone fragment.

8. REFERENCES

[1] A. Alali, B. Bartman, C. D. Newman, J. I. Maletic, "A Preliminary Investigation of Using Age and Distance Measures in the Detection of Evolutionary Couplings", Proc. *MSR*, 2013, pp. 169 – 172.

[2] L. Aversano, L. Cerulo, and M. D. Penta, "How clones are maintained: An empirical study", Proc. *CSMR*, 2007, pp. 81-90.

[3] L. Barbour, F. Khomh, Y. Zou, "Late Propagation in Software Clones", Proc. *ICSM*, 2011, pp. 273 – 282.

[4] J .R. Cordy and C.K. Roy, "The NiCad Clone Detector", Proc *ICPC Tool Demo*, 2011, pp. 219 – 220.

[5] Cross Cutting Concerns: `http://en.wikipedia.org/wiki/Cross-cutting_concern`

[6] E. Duala-Ekoko, and M. P. Robillard, "CloneTracker: Tool Support for Code Clone Management", Proc. *ICSE*, 2008, pp. 843 – 846.

[7] E. Duala-Ekoko, and M. P. Robillard, "Tracking Code Clones in Evolving Software", Proc. *ICSE*, 2007, pp. 158 - 167.

[8] H. Gall, K. Hajek, and M. Jazayeri, "Detection of logical coupling based on product release history," Proc. *ICSM*, 1998, pp. 190–199.

[9] N. Göde, Rainer Koschke, "Frequency and risks of changes to clones.", Proc. *ICSE*, 2011, pp. 311 – 320.

[10] N. Göde, J. Harder, "Clone Stability", Proc. *CSMR*, 2011, pp. 65-74.

[11] P. Jablonski, and D. Hou, "CReN: A tool for tracking copy-and-paste code clones and renaming identifiers consistently in the IDE." Proc. *Eclipse Technology Exchange at OOPSLA*, 2007.

[12] C. Kapser and M. W. Godfrey, ""Cloning considered harmful" considered harmful: patterns of cloning in software", *ESE*, 13(6), 2008, pp. 645-692.

[13] M. Kim, V. Sazawal, D. Notkin, and G. C. Murphy, "An empirical study of code clone genealogies", Proc. *ESEC-FSE*, 2005, pp. 187-196.

[14] J. Krinke, "A study of consistent and inconsistent changes to code clones", Proc. *WCRE*, 2007, pp. 170-178.

[15] J. Krinke, "Is cloned code more stable than non-cloned code?", Proc. *SCAM*, 2008, pp. 57-66.

[16] A. Lozano and M. Wermelinger, "Tracking clones' imprint", Proc. *IWSC*, 2010, pp. 65-72.

[17] A. Lozano, and M. Wermelinger, "Assessing the effect of clones on changeability", Proc. *ICSM*, 2008, pp. 227-236.

[18] R. C. Miller, and B. A.Myers. "Interactive simultaneous editing of multiple text regions.", Proc. *USENIX 2001 Annual Technical Conference*, 2001, pp. 161 – 174.

[19] M. Mondal, C. K. Roy, and K. A. Schneider, "Connectivity of Co-changed Method Groups: A Case Study on Open Source Systems", Proc. *CASCON*, 2012, pp. 205-219.

[20] M. Mondal, C. K. Roy, and K. A. Schneider, "Automatic Ranking of Clones for Refactoring through Mining Association Rules", Proc. *CSMR-WCRE*, 2014, 10 pp. (to appear)

[21] M. Mondal, C. K. Roy, and K. A. Schneider, "Comparative Stability of Cloned and Non-cloned Code: An Empirical Study", Proc. *SAC*, 2012, pp. 1227 – 1234

[22] M. Mondal, C. K. Roy, and K. A. Schneider, "An Empirical Study on Clone Stability", *ACM SIGAPP Applied Computing Review*, 2012, 12(3): 20 – 36.

[23] C. K. Roy, "Detection and analysis of near-miss software clones.", Proc. *ICSM*, 2009, pp. 447 – 450.

[24] C. K. Roy and J. R. Cordy, "NICAD: Accurate Detection of Near-Miss Intentional Clones Using Flexible Pretty-Printing and Code Normalization", Proc. *ICPC*, 2008, pp. 172 – 181.

[25] C. K. Roy, J.R. Cordy and R. Koschke, "Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach", *SCP*, 2009, 74 (2009): 470 – 495.

[26] C.K. Roy, J.R. Cordy, "A Mutation / Injection-based Automatic Framework for Evaluating Code Clone Detection Tools", Proc. *Mutation*, 2009, pp. 157 – 166.

[27] M. J. Shepperd, S. G. MacDonell, "Evaluating prediction systems in software project estimation". *Information & Software Technology*, 2012, 54(8): 820 – 827.

[28] S. Thummalapenta, L. Cerulo, L. Aversano, and M. D. Penta, "An empirical study on the maintenance of source code clones", *ESE*, 15(1), 2009, pp. 1-34.

[29] M. Toomim, A. Begel, and S. L. Graham. "Managing duplicated code with linked editing.", Proc. *IEEE Symposium on Visual Languages and Human Centric Computing*, 2004, pp. 173 – 180.

[30] T. Wang, M. Harman, Y. Jia, J. Krinke, "Searching for Better Configurations: A Rigorous Approach to Clone Evaluation", Proc. *ESEC/SIGSOFT FSE*, 2013, pp. 455 – 465.

[31] T. Zimmermann, P. Weisgerber, S. Diehl, A. Zeller, "Mining version histories to guide software changes," Proc. *ICSE*, 2004, pp. 563–572.