An Empirical Study on Ranking Change Recommendations Retrieved using Code Similarity

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

Abstract—**Providing change suggestions for a particular code snippet on the basis of how similar code snippets were changed in the past has been investigated by a number of studies. These studies rank change recommendations emphasizing their frequency of occurrence during the prior evolution. In our study, we investigate the ranking of change recommendations on the basis of their recency of occurrence in the past and compare this technique with the frequency based technique. According to our experimental results on thousands of revisions of six subject systems we observe that while ranking on the basis of frequency performs better than recency based ranking, a combination of these two techniques performs significantly better than the discrete techniques. We find that the combined technique provides better overall rankings for 16.58% more cases when compared with the frequency ranking technique, and 57.48% more cases when compared with the recency ranking technique.**

I. Introduction

A number of existing studies [13], [19], [20], [24] have investigated the possibility of providing change suggestions to programmers when they attempt to make some changes to the source code. The main idea behind these existing approaches is to infer change suggestions from the evolution history of the software systems. Let us assume that a programmer is currently going to make some changes to a code snippet. If it is observed that a similar code snippet was previously changed during the past evolution, then the change that occurred to this similar code snippet can be suggested to the programmer for changing the current code snippet (i.e., the target snippet).

There can be multiple cases in the past where a similar code snippet (i.e., similar to the target code snippet) was changed. Thus, we can get multiple change suggestions for changing a particular target snippet. However, the programmer will choose only one of these suggestions. Thus, it is important to order the change suggestions using an efficient ranking mechanism so that the most promising suggestion gets a high rank.

The existing approaches [20], [24] rank the change suggestions on the basis of their occurrence frequencies in the past. Thus, in the existing approaches, if a particular suggested change occurred more frequently in the past compared to the other suggestions, then that particular change suggestion gets a higher priority compared to the others. However, change suggestions can also be ranked by considering their occurrence recency, where a change suggestion that occurred more recently can be given a higher priority compared to others. In a previous study [14] we used recency ranking for ranking co-change candidates of code clones.

In our research we investigate both recency ranking and frequency ranking for ranking change recommendations. We see that recency ranking sometimes performs better than frequency ranking. We propose a hybrid ranking mechanism combining these two. According to our in-depth investigation on thousands of commits of six diverse subject systems written in two programming languages:

(1) While frequency ranking appears to be better than recency ranking in general, our proposed hybrid ranking mechanism (i.e., a ranking mechanism that combines both frequency and recency ranking) performs significantly better than each individual one according to our statistical significance tests;

(2) Hybrid ranking provides better ranks for an overall 16.58% more cases when compared with frequency ranking, and 57.48% more cases when compared with recency ranking.

The rest of the paper is organized as follows. Section II defines the frequency and recency ranking mechanisms, Section III mentions the experimental steps, Section IV compares frequency and recency ranking mechanisms, in Section V we propose a hybrid ranking mechanism and investigate its ranking efficiency in comparison with frequency ranking and recency ranking, Section VI discusses the related work, Section VII mentions possible threats to validity, and Section VIII concludes the paper mentioning future work.

II. Terminology

Frequency Ranking of change recommendations. Let us assume that a change recommendation system has inferred a number of change suggestions for changing a target code snippet by analyzing the previous commits. It is possible that two or more change suggestions retrieved from the same or different commits are the same. We first identify the distinct change suggestions. For each of these distinct suggestions we determine the number of times it occurred in the past. We call this number the frequency of a change suggestion. Then, we sort these distinct suggestions in decreasing order of their occurrence frequencies. The change suggestions with higher frequencies are given higher priorities [20], [24].

Recency Ranking of change recommendations. In this technique, we first determine the distinct change suggestions. For each distinct suggestion we determine its recency. A suggested change (i.e., a change suggestion) might occurr in multiple commits. The recency of a suggested change is the latest commit operation where it occurred. Finally, we sort the distinct suggestions in decreasing order of their recency. In this technique, the change suggestions with higher recency (i.e., the change suggestions that occurred more recently) are given higher priorities.

Examples of Frequency and Recency Ranking. Table I shows the ranking of seven distinct change suggestions, S1 to S7. We can see the frequency as well as the recency of each suggestion. In the frequency based ordering we see that the suggestions: S5, S6, and S7 with the highest frequency (i.e., 4) come first (i.e., at the left). In case of recency based ordering, the most recent suggestions (i.e., S2, and S4) have been given the highest priorities.

In Section V we propose a hybrid ranking mechanism combining frequency ranking and recency ranking mechanisms. We show that our proposed hybrid ranking mechanism performs significantly better than each individual one.

TABLE I: Different ways of ranking change suggestions

	S1	S2	S3	S4	S5	S6	S7
Frequency	2	2	1	3	4	4	4
Recency	101	190	103	190	98	101	74
Frequency based ordering	S5	S6	S7	S4	S1	S2	S3
Recency based ordering	S2	S4	S3	S1	S6	S5	S7

S1 to S7 are the distinct change suggestions.
Recency is the most recent commit operation where a change suggestion occurred.

TABLE II: Subject Systems

Systems	Lang.	Domains	LLR	Revisions
jEdit	Java	Text Editor	191,804	4000
Freecol	Java	Game	91,626	1950
Carol	Java	Game	25,091	1700
Jabref	Java	Reference Management	45,515	1545
Camellia	C	Image Processing Library	88,033	170
JHotDraw	Java	Graphics	143,339	799

LLR = LOC in Last Revision

III. Experimental Steps

We perform our investigation on six subject systems listed in Table II. We download these systems from an open-source SVN repository[1]. For each of the subject systems we perform the following steps: **(1)** Download all the revisions of the subject system as mentioned in Table II, **(2)** Preprocess the source code files in each revision by removing comments, blank lines, and indentations, **(3)** Extract changes between every two consecutive revisions using *diff*, **(4)** Build a change suggestion database from the extracted changes by applying our change suggestion technique developed in our earlier study [13], and **(5)** investigate and compare the ranking mechanisms.

Change Suggestion Technique. Our change suggestion technique [13] works in the following way. Let us assume that a programmer is going to make some changes to a target code snippet CS_{target} in the code-base. Our technique automatically examines the previous commits to determine whether any similar code snippet $CS_{previous}$ was changed in a previous commit operation. Our technique suggests the change that occurred to the similar code snippet $CS_{previous}$ for the target code snippet CS_{target}. Our technique [13] can provide change suggestions with a precision of up to 23% which is reasonable with respect to the other existing studies [20], [24]. Using our technique we can provide change suggestions for up to 30.47% of the cases.

Our change suggestion technique works on the exact similarity of the code snippets. If the target code snippet CS_{target} is exactly similar to the code snippet $CS_{previous}$ that was previously changed, then our technique extracts change suggestion from $CS_{previous}$. We consider exact similarity in our research because the retrieved suggestions can be readily applied to the target snippet. Suggestions can also be retrieved by considering syntactic similarity. However, in this case the suggestions might be just templates and cannot be readily applied to the target code snippet [24]. In future we will investigate ranking of change suggestions considering syntactic similarity. For the details of our change suggestion technique we refer the interested readers to our earlier work [13].

Similarity Detection. We detect exact similarity using an adapted version of NiCad [3], [27] technology where we can detect the similarity of two given code snippets of any granularity (i.e., block or method). We use NiCad because it is capable of detecting clones with high precision and recall [25]–[27].

[1]Open-Source SVN Repository: http://www.sourceforge.net

TABLE III: Statistics regarding change suggestions

Systems	jEdit	Freecol	Carol	Jabref	Camellia	jHotDraw
TNC	6235	16308	8241	14901	3115	60853
NCS	1187	4969	1573	2630	445	12354
NCCS	200	753	370	375	65	6517

TNC = Total number of changes during system evolution.
NCS = No. of cases where our technique can provide change suggestion(s)
NCCS = No. of cases where the list of suggestions provided by our technique contains the correct change suggestion to be applied.

Building a change suggestion database. We build a change suggestion database for our study by using our change suggestion technique [13]. Let us assume that a particular code snippet CS_R residing in revision R was changed in the commit operation C applied on R. The corresponding snippet in the next revision $R+1$ is CS_{R+1}. Our change suggestion technique examines the previous commits (i.e., the commits from 1 to $C-1$) to identify cases where a similar code snippet (i.e., similar to CS_R) was previously changed. From these cases our technique extracts change suggestions for changing the particular code snippet CS_R. However, by comparing the code snippets, CS_R and CS_{R+1}, we can determine which change was actually applied to CS_R. In the change suggestion database we include the followings: (1) the particular code snippet CS_R, (2) the change suggestions retrieved by our technique for changing CS_R, and (3) the actual change which was applied to CS_R. We can easily understand that if the actual change which was applied on CS_R is present in the list of change suggestions retrieved by our technique, then we can say that the change suggestion list contains the correct suggestion. Table III shows statistics regarding change suggestions in our subject systems. In our study we investigate which ranking mechanism provides a better rank to the correct suggestion.

IV. Comparing Frequency and Recency Ranking

In this section we make a comparison between the two ranking mechanisms: *Frequency Ranking* and *Recency Ranking*. We compare their efficiencies in ranking the correct change suggestion (i.e., the actual change to be implemented) among all the change suggestions retrieved by our change suggestion mechanism. First, we examine all the commit operations of a particular subject system to identify those cases that we analyze for comparing the ranking mechanisms. We call these cases the *eligible cases*. We define a case and an eligible case in the following way.

A Case. *A case consists of two things: (1) a target code snippet which a programmer wants to change, and (2) a list of change suggestions inferred by our mechanism for changing that target snippet. If our change suggestion technique cannot infer any change suggestions for the target code snippet, then the list of suggestions will be empty.*

An Eligible Case. *An eligible case is that case where the list of change suggestions is not empty and contains the actual change to be applied to the target code snippet. We call this actual change to be applied to the target code snippet the* ***correct change suggestion****. Thus, an eligible case is the case where the list of change suggestions provided by our technique contains the correct change suggestion. We identify the eligible cases in the following way.*

A. Identification of the Eligible Cases

We examine each of the commit operations starting from the very beginning one as mentioned in Table II. Let us assume that we are currently examining the commit operation C. A code snippet $s_{current}$ experienced the change $c_{current}$ in this commit operation. Now, we apply our change suggestion technique to determine a list of change suggestions from the previous commits 1 to C-1 for changing $s_{current}$. If a suggested change $c_{suggested}$ in this list is exactly similar to $c_{current}$, then $c_{suggested}$ is the correct change suggestion, and this case consisting of $s_{current}$ and the list of suggested changes is an eligible case. As we examine each of the commit operations from the beginning one, we identify all the eligible cases. We investigate these eligible cases for comparing the ranking mechanisms. A case which is not an eligible one cannot be used for investigating ranking because the corresponding list of change suggestions will either be empty or will not contain the *correct change suggestion*. Table III shows the number of eligible cases (the row **NCCS**) for each of our subject systems.

B. Comparing ranking mechanisms using eligible cases

Let us consider two ranking mechanisms, RM1 and RM2, and an eligible case EC. We use each of these two ranking mechanisms to rank or order the change suggestions in the suggestion list of EC. We can easily understand that the ranking mechanism that assigns a better rank to the correct suggestion in the list should be considered the superior one.

Let us assume that there are six change suggestions: S1, S2, S3, S4, S5, and S6 in the suggestion list of the eligible case EC. These suggestions were inferred by our change suggestion mechanism. Let us also assume that the correct change suggestion is S4. We show these six change suggestions in Fig. 1, and also, identify the correct suggestion S4. We use each of the two ranking mechanisms RM1 and RM2 to rank or order the six suggestions. These orderings are also shown in the figure. From the figure we see that the serial number of the correct change suggestion in the ordering regarding RM1 is 5. The corresponding serial number in case of RM2 is 2. We understand that RM2 assigns a better rank (i.e., a lower serial number) to the correct change suggestion compared to RM1. Thus, for this example eligible case (i.e., EC), RM2 is better than RM1. Considering each of the eligible cases of a particular subject system we compare the frequency ranking and recency ranking mechanisms.

C. Comparing Frequency Ranking and Recency Ranking

We examine all the eligible cases of a particular subject system considering both frequency ranking and recency ranking and determine the following measures. We report these measures in Table IV.

Measure 1 (the column 'F $>$ R' in Table IV): The number of cases where frequency ranking assigns a better rank to the correct change suggestion compared to recency ranking.

Measure 2 (R $>$ F): The number of cases where recency ranking assigns a better rank to the correct change suggestion compared to frequency ranking.

Measure 3 (F = R): The number of cases where both mechanisms provide the same rank to the correct suggestion.

Measure 4 (Favg): The average of the ranks assigned to the correct suggestions in case of frequency ranking.

Measure 5 (Ravg): The average of the ranks assigned to the correct suggestions in case of recency ranking.

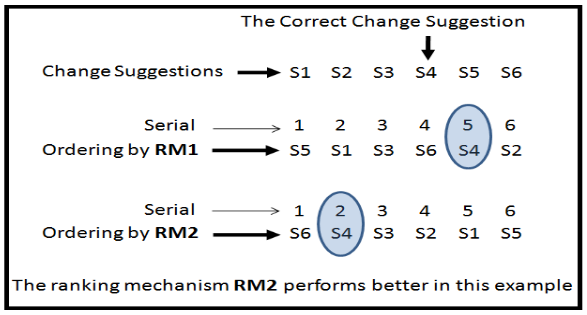

Fig. 1: Comparing the ranking mechanisms, RM1 and RM2

TABLE IV: Comparing Frequency and Recency Ranking

Systems	NECS	F $>$ R	R $>$ F	F = R	Favg	Ravg
jEdit	200	61	18	121	13.63	11.98
Freecol	753	233	127	393	5.88	8.16
Carol	370	145	70	155	11.09	7.86
Jabref	375	191	81	103	6.49	23.85
JHotDraw	6517	3302	1443	1772	4.43	8.98
Camellia	65	9	14	42	4.21	3.88

NECS = No. of eligible cases (i.e., the cases with correct suggestion)
F $>$ R = The no. of cases where frequency ranking provides better rank.
R $>$ F = The no. of cases where recency ranking provides better rank.
'F = R' is the no. of cases where both mechanisms provide the same rank.
Favg = Avg. rank of the correct suggestions considering frequency ranking
Ravg = Avg. rank of the correct suggestions considering recency ranking

We also determine the following two percentages from the two measures, **Measure 1** and **Measure 2**, defined above.

Percentage 1 (%F$>$R): The percentage of the cases where frequency ranking assigns a better rank to the correct change suggestion compared to recency ranking with respect to all eligible cases. We determine this percentage from **Measure 1** (in the list of five measures above) in the following way.

$$Percentage\ 1 = (Measure\ 1\ /\ NECS) \times 100 \qquad (1)$$

Here, NECS (reported in Table IV) is the total number of eligible cases of a particular subject system.

Percentage 2 (%R$>$F): The percentage of the cases where recency ranking assigns a higher rank to the correct change suggestion compared to frequency ranking with respect to all eligible cases. We determine this percentage from **Measure 2** using an equation similar to Eq. 2.

We show these percentages in the bar graph of Fig. 2 to make a visual comparison of the efficiencies of the two ranking mechanisms: frequency ranking and recency ranking.

Comparison 1. From Fig. 2 we see that for five out of six subject systems (i.e., except Camellia), the percentage of cases where frequency ranking provides better ranks to the correct change suggestions is greater than the percentage of cases where recency ranking provides better ranks to the correct suggestions. In other words frequency ranking exhibits overall better performance compared to recency ranking.

Statistical Significance Test Regarding Comparison 1. We perform Mann-Whitney-Wilcoxon (MWW) test [12] to determine whether the six percentages (i.e., corresponding to six subject systems) regarding frequency ranking are significantly different than the percentages regarding recency ranking. MWW test is a non-parametric test and does not require the samples to be normally distributed [21]. This test can be applied to both large and small samples [11]. We perform

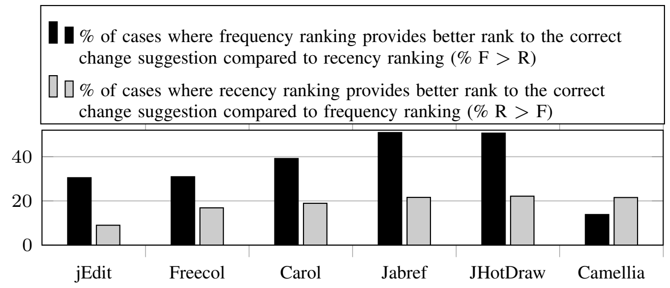

Fig. 2: Comparing frequency ranking and recency ranking.

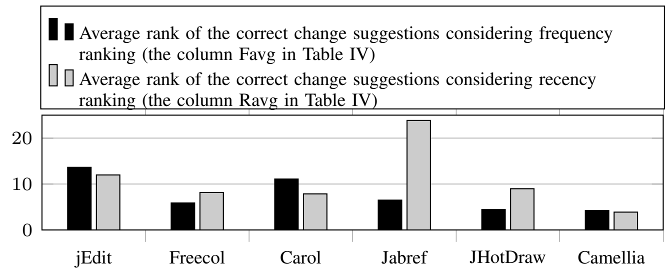

Fig. 3: Comparing the average ranks of the correct suggestions provided by frequency ranking and recency ranking.

the test considering a significance level of 5%. According to our tests, the percentages regarding frequency ranking are significantly different than the percentages regarding recency ranking with a *p-value* of 0.04 for two-tailed test and 0.02 for one-tailed test, and with an effect size [6] of 0.6. The effect size calculation procedure for MWW test is available on-line [7]. We see that the *p-values* are less than 0.05. We finally understand that *the percentage of cases where frequency ranking provides better ranks to the correct change suggestions is significantly higher compared to the corresponding percentage of cases where recency ranking provides better ranks.*

Comparison 2. We draw a graph (Fig. 3) showing the comparison of the average ranks of the correct change suggestions provided by frequency ranking and recency ranking. From the graph we see that for three subject systems (i.e., jEdit, Carol, and Camellia) out of six, the average rank of the correct change suggestions considering recency ranking is better than that of frequency ranking (i.e., the smaller a rank the better it is, because rank is the serial number of the correct change suggestion provided by a particular ranking mechanism).

Statistical Significance Test Regarding Comparison 2. We perform MWW test [11], [21] to determine whether the average ranks calculated considering frequency ranking are significantly different than the average ranks calculated considering recency ranking. According to our tests performed considering a significance level of 5%, the average ranks measured using frequency ranking are not significantly different than those measured using recency ranking. The *p-values* for the two-tailed and one-tailed test cases are 0.48 and 0.24 respectively, and the effect size is 0.23.

Discussion. The existing studies [20], [24] considered frequency ranking for ordering/ranking the change suggestions. From our first comparison (i.e., Comparison 1) we see that their consideration is reasonable compared to recency ranking. However, although frequency ranking appears to be better than recency ranking we see (from Table IV as well as from Fig. 2) that for many cases of each of the subject systems recency ranking provides better ranks to the correct change suggestions compared to frequency ranking. For example, if we consider our subject system Freecol we see (from Table IV) that for 127 of the eligible cases recency ranking provides better ranks to the correct change suggestions compared to frequency ranking. Moreover, from our second comparison (i.e., Comparison 2) we realize that the average ranks of the correct change suggestions calculated using frequency ranking are not significantly different than the average ranks calculated using recency ranking. Such a scenario implies that a more recent change suggestion often gets prioritized. Thus, a combination of these two ranking mechanisms might possibly help us achieve an even better ranking of the change suggestions compared to both frequency ranking and recency ranking. Considering this issue, in the following section we propose a hybrid ranking mechanism that combines both frequency ranking and recency ranking.

V. Proposing a Hybrid Ranking Mechanism

From our investigation in the Section IV we found that frequency ranking is a better choice than recency ranking in general for ranking change suggestions. However, there can be multiple change suggestions with the same frequency. We did not apply any particular ranking mechanism for such same frequency suggestions. In the following subsection we investigate which change suggestion generally get prioritized in presence of multiple change suggestions having the same frequency. Such an investigation can help us choose a particular ranking mechanism for the same frequency suggestions.

A. Investigation on Ranking Change Suggestions having the Same Frequency

In this subsection we analyze whether the most recent one gets prioritized when there are multiple change suggestions with the same frequency. By examining the commit operations of a subject system we select particular *eligible cases* for our investigation. We defined *eligible case* in Section IV. Let us consider an eligible case EC where the correct change suggestion has a frequency of f. We consider this case for our investigation if it satisfies the following two conditions.

Condition 1. There is at least one more change suggestion with the frequency of f in the suggestion list of EC.

Condition 2. All the change suggestions with the frequency of f should not have the same recency. Recency is the last commit operation where a suggested change occurred.

The first condition ensures that there are multiple change suggestions with the same frequency and one of these suggestions is the correct suggestion. Also, in the absence of the second condition it might happen that all the change suggestions with the frequency of f have the same recency. We can easily understand that the above two conditions are necessary for a case to be considered for our investigation. For a particular subject system we first identify all the cases each satisfying the above two conditions, and then separate these cases into two disjoint sets as described below:

CMR: This set contains those cases where the correct change suggestion is the most recent one.

CNMR: This set contains those cases where the correct change suggestion is not the most recent one.

Table V shows the total number of cases each satisfying the above two conditions, percentage of such cases with respect to all *eligible cases*, and also, the number of cases in each of the two sets, **CMR** and **CNMR**, for each subject system. We also determine the percentage of cases in each of the two

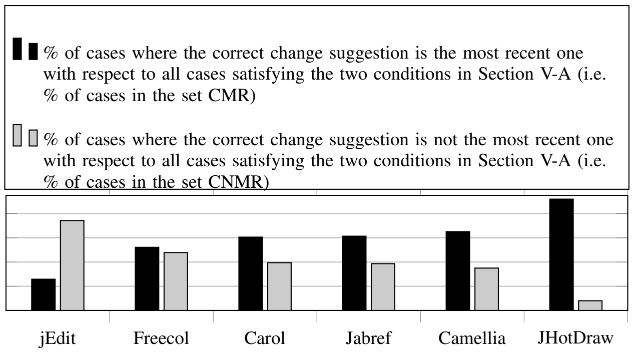

Fig. 4: Statistics of the same frequency suggestions.

TABLE V: Statistics regarding same frequency suggestions

Systems	jEdit	Freecol	Carol	Jabref	Camellia	jHotDraw
NCSC	31	182	99	101	20	1453
CMR	8	95	60	62	13	1337
CNMR	23	87	39	39	7	116
NECS	200	753	370	375	65	6517
PC	15.5%	24.2%	26.8%	26.9%	30.8%	22.3%

NCSC = Total no. of cases each satisfying the two conditions in Section V-A.
= CMR + CNMR
CMR = No. of cases where the correct suggestion is the most recent one.
CNMR = No. of cases where the correct suggestion is not the most recent one.
NECS = No. of eligible cases (i.e., the cases with correct suggestion)
PC = NCSC * 100 / NECS = % of cases each satisfying the two conditions in Section V-A with respect to all *eligible cases* (i.e., NECS).

sets, **CMR** and **CNMR**, with respect to all the cases satisfying the above two conditions. We show these percentages in Fig. 4. The figure shows that for most of the subject systems except jEdit, the percentage of cases where the correct change suggestion is the most recent one is higher than the percentage of cases where the correct change suggestion is not the most recent one. From this we realize that *generally the most recent suggestion gets prioritized when multiple change suggestions have the same frequency. Thus, we can possibly consider recency ranking technique for ranking change suggestions having the same frequency.*

B. Hybrid Ranking Mechanism

On the basis of our findings in Section IV-C and V-A we propose a hybrid ranking mechanism for ranking change suggestions. In hybrid ranking technique we order the change suggestions in two steps. In the first step, we sort the distinct change suggestions in decreasing order of their occurrence frequencies. In the second step, we identify groups of the distinct change suggestions where all the suggestions in a particular group have the same frequency. We sort the suggestions in each such group in decreasing order of their occurrence recency as described in Section II.

Table VI shows the same set of distinct change suggestions that we used (in Table I) for demonstrating frequency and recency ranking in Section II. In Table VI we show the two steps of hybrid ranking. In the first step, the suggestions S1 to S7 are ordered in decreasing order of their frequencies. We can easily understand that after this step, the suggestions with the same frequency become grouped together. For example, in Step 1 in Table VI we see that S5, S6, and S7 have the same frequency of 4 and they are adjacent to one another. We consider them (i.e., S5, S6, and S7) a group. Also, S1 and S2 form another group. In Step 2, the suggestions in each of these two groups have been reordered in decreasing order of their

TABLE VI: Example of Hybrid Ranking

	S1	S2	S3	S4	S5	S6	S7
Frequency	2	2	1	3	4	4	4
Recency	101	190	103	190	98	101	74
Step 1	S5	S6	S7	S4	S1	S2	S3
Step 2	S6	S5	S7	S4	S2	S1	S3

S1 to S7 are the distinct change suggestions.
Recency is the most recent commit of occurrence of a suggestion.

TABLE VII: Comparing Hybrid and Frequency Ranking

Systems	NECS	H > F	F > H	H = F	Havg	Favg
jEdit	200	18	14	168	8	13.63
Freecol	753	100	61	592	4.22	5.88
Carol	370	64	31	275	3.76	11.09
Jabref	375	77	25	273	3.17	6.49
JHotDraw	6517	1339	103	5075	3.45	4.43
Camellia	65	11	2	52	2.92	4.21

NECS = No. of eligible cases (i.e., the cases with correct suggestion)
H > F = The no. of cases where hybrid ranking provides better rank.
F > H = The no. of cases where frequency ranking provides better rank.
'H = F' is the no. of cases where both mechanisms provide the same rank.
Havg = Avg. rank of the correct suggestions considering hybrid ranking
Favg = Avg. rank of the correct suggestions considering frequency ranking

recency. In the following subsections we investigate whether our proposed hybrid ranking is better than frequency ranking and recency ranking.

C. Comparing Hybrid Ranking and Frequency Ranking

We compare hybrid ranking and frequency ranking following the same procedure that we followed (in Section IV-C) to make a comparison between frequency and recency ranking. Table VII shows the five measures regarding the comparison.

Comparison 1. We draw Fig. 5 for showing a visual comparison of the efficiencies of the two ranking mechanisms. From Fig. 5 we see that for each of the subject systems hybrid ranking performs better than frequency ranking. The percentage of cases where hybrid ranking provides better ranks to the correct change suggestions is always higher than the percentage of cases where frequency ranking provides better ranks to the correct change suggestions. Considering all subject systems we see that hybrid ranking provides better ranks than frequency ranking for an overall 19.43% of the eligible cases. We also see that for overall 2.85% of the eligible cases frequency ranking provides better ranks than hybrid ranking. Thus, we see that hybrid ranking provides better ranks to the correct change suggestions than frequency ranking for an overall 16.58% (19.43% - 2.85%) more cases.

Statistical significance test regarding Comparison 1. We perform the Mann-Whitney-Wilcoxon test [11], [21] considering a significance level of 5% to determine whether the six percentages regarding hybrid ranking are significantly higher compared to those regarding frequency ranking. According to our tests, the six percentages (i.e., the samples) regarding hybrid ranking are significantly different than those of frequency ranking. The *p-value* for both two-tailed and one-tailed test cases is less than 0.01, and the effect size (0.83) is large. Thus, *the percentages regarding hybrid ranking are significantly higher than those regarding frequency ranking.*

Comparison 2. In Fig. 6 we show the comparison of the average ranks of the correct change suggestions provided by hybrid ranking and frequency ranking. From this figure we realize that the average rank of the correct change suggestions

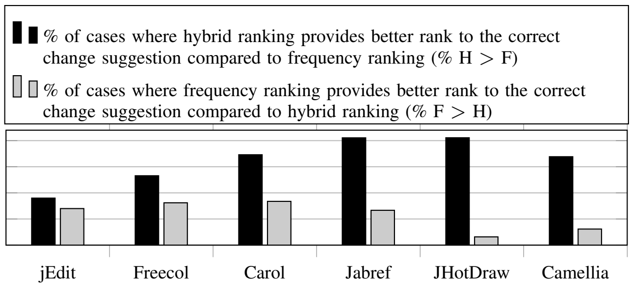

Fig. 5: Comparing hybrid ranking and frequency ranking.

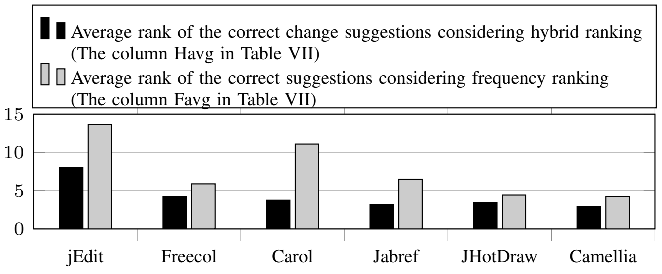

Fig. 6: Comparing the average ranks of the correct suggestions provided by hybrid ranking and frequency ranking.

considering hybrid ranking is always better than the average rank of the correct change suggestions considering frequency ranking. We should again note that the lower a rank the better it is, because a rank is the serial number/position value of a change suggestion in the list of all change suggestions.

Statistical significance test regarding Comparison 2. Again, we performed MWW test to determine whether the six average ranks for six subject systems calculated using hybrid ranking are significantly different than the six average ranks calculated using frequency ranking. We perform the test considering a significance level of 5%. According to our tests, the average ranks provided by hybrid ranking are significantly different than those regarding frequency ranking. The *p-values* for the two-tailed and one-tailed test cases are 0.04 and 0.02 respectively, and the effect size is 0.6. We see that the *p-values* are less than 0.05. As the average ranks provided by hybrid ranking are smaller (i.e., better) than the corresponding ranks provided by frequency ranking, we can state that *the average ranks of the correct change suggestions calculated considering hybrid ranking are significantly better compared to the average ranks obtained considering frequency ranking*.

Discussion. From our comparison between frequency ranking and recency ranking it appeared that frequency ranking performs better than recency ranking in case of Comparison 1 (i.e., comparison regarding the percentage of cases where the ranking mechanisms perform better). However, this was not true in case of Comparison 2. We found that the average ranks of the correct change suggestions calculated using frequency ranking are not significantly better than those calculated using recency ranking. Interestingly, from our comparison between hybrid ranking and frequency ranking we see that hybrid ranking performs significantly better than frequency ranking in both ways (i.e, both in Comparison 1 and Comparison 2). From this we infer that hybrid ranking is a better choice than frequency ranking while ranking the change recommendations. Hybrid ranking not only provides better ranks than frequency ranking for a significantly higher number of eligible cases but also, the average ranks of the correct change suggestions calculated using hybrid ranking are significantly better than those calculated using frequency ranking.

We also compared hybrid ranking and recency ranking mechanisms and found that hybrid ranking performs significantly better than recency ranking both in Comparison 1 and Comparison 2 (described above). According to our calculation, hybrid ranking provides better ranks to the correct change suggestions than recency ranking for overall 57.48% more cases. We finally state that *hybrid ranking is the best choice among the three mechanisms (i.e., frequency ranking, recency ranking, and hybrid ranking) for ranking change suggestions*.

VI. Related Work

Recommending changes by analyzing the past evolution history of a software system is not new. Our focus in this study is however the ranking of change recommendations.

In two previous studies Nguyen et al. [20], and Ray et al. [24] implemented and investigated change recommendation systems. They ranked change suggestions only on the basis of their occurrence frequencies. In another study Nguyen et al. [19] investigated recurring bug-fixes as well as recurring changes. They implemented a system to identify code peers and to automatically suggest changes to a code fragment where the changes were experienced by a peer code fragment. They did not apply any particular ranking mechanism for ranking change suggestions. We previously conducted an empirical study [13] on change recommendations. However, we also did not apply any particular ranking mechanism for ranking recommendations. In the study presented in this paper we investigate both frequency ranking and recency ranking, and finally propose a hybrid ranking mechanism combining these two. We show that our proposed hybrid ranking mechanism performs significantly better than each individual one.

We performed an empirical study [16] for identifying the important code clones for refactoring. We applied frequency ranking technique for ranking the refactoring candidates. The code clones that changed together more frequently were prioritized for refactoring. Tsantalis and Chatzigeorgio [31] ranked refactoring suggestions considering recency ranking. There are also a number of studies [4], [10], [15], [34] that recommend peer code artifacts for co-changing (i.e., changing together) while changing a particular artifact on the basis of the past evolution history. These studies mainly emphasize frequency ranking technique for ranking of co-change candidates. In our study we investigate the ranking of change recommendations considering both frequency and recency ranking, and propose a hybrid one that performs better than each individual one.

Toomin et al. [30] performed a study on simultaneous editing of multiple clone fragments in the working codebase of a software system. CloneTracker [5] also supports simultaneous editing of the clone fragments tracked by it. While these studies mainly deal with propagating a change that occurred in one clone fragment to a peer clone fragment, they cannot infer which changes might occur in a particular code fragment. Moreover, they do not deal with any ranking mechanism. Our study, on the other hand, deals with inferring change suggestions for any code fragment whether it is a clone fragment or not. Also, the primary goal of our study is the ranking of the inferred change suggestions.

A number of studies [1], [2], [8], [9], [17], [18], [22], [23], [28], [33] have also been done on code completion, particularly on method call completion or method body completion. Most

[2], [9], [23], [33] of these studies used frequency ranking technique for ranking of suggestions. Some studies [17], [18] also used graph based techniques and context sensitivity [1]. Robbes and Lanza [28] performed recency ranking of suggestions. Our investigation involves recommending possible changes to any target code snippet by retrieving and analyzing the changes that occurred to similar code snippets in the past. Our primary goal is the ranking of change recommendations. We investigate both frequency ranking and recency ranking for ranking change suggestions. We also propose a hybrid ranking mechanism combining these two and show that the hybrid one performs significantly better than each individual one.

We believe that our proposed hybrid ranking mechanism can continuously help programmers during development and maintenance. According to our experimental results and anlaysis, hybrid ranking is the best choice for ranking change recommendations among our investigated ranking techniques.

VII. Threats to Validity

We use the NiCad clone detector [3] in our experiment. While clone detectors suffer from *confounding configuration choice problem* [32], NiCad has been found to perform fairly well [26], [27]. Also, in a recent study [29] Svajlenko and Roy show that NiCad is a very good choice for detecting clones compared to other modern clone detectors.

The subject systems studied in this research are not enough to make a concrete decision about ranking change suggestions. However, our subject systems are of diverse variety in terms of application domains, system size (LOC), and the number of revisions. Thus, we believe that the outcome of our study cannot be attributed to a chance, and our findings are of significant importance for ranking change recommendations.

VIII. Conclusion

In this research we perform an empirical study on ranking change recommendations retrieved on the basis of exact similarity of code fragments. We investigate and compare two preexisting ranking mechanisms, *frequency ranking* and *recency ranking*, and finally propose a hybrid ranking mechanism by reasonably combining the strengths of these two. According to our investigation on thousands of commits of six diverse subject systems written in two programming languages:

(1) Our proposed hybrid ranking mechanism performs significantly better (according to our statistical significance tests) compared to each of the other two techniques.

(2) Hybrid ranking provides better ranks to the correct change suggestions for around 16.58% more cases when compared with frequency ranking, and 57.48% more cases when compared with recency ranking.

In future we would like to investigate the efficiency of the hybrid ranking mechanism in different other contexts such as ranking of code clones for refactoring and tracking, and ranking of co-change candidates for different program entities.

References

[1] M. Asaduzzaman, C. K. Roy, K. Schneider, D. Hou, “CSCC: Simple, Efficient, Context Sensitive Code Completion”, Proc. *ICSME*, 2014, pp. 71 – 80.

[2] M. Bruch, M. Monperrus, M. Mezini, “Learning from examples to improve code completion systems”, Proc. *FSE*, 2009, pp. 213 – 222.

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

[4] E. Duala-Ekoko, M. P. Robillard,“Tracking Code Clones in Evolving Software”, Proc. *ICSE*, 2007, pp. 158 – 167.

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

[6] Effect Size: http://en.wikipedia.org/wiki/Effect_size

[7] Effect Size Calculation for Mann-Whitney-Wilcoxon Test: http://www.let.rug.nl/~heeringa/statistics/stat03_2013/lect09.pdf

[8] R. Hill, J. Rideout, “Automatic method completion”, Proc. *ASE*, 2004, pp. 228 – 235.

[9] D. Hou, D. M. Pletcher, “An evaluation of the strategies of sorting, filtering, and grouping API methods for Code Completion”, Proc. *ICSM*, 2011, pp. 233 – 242.

[10] H. Kagdi, M. Gethers, D. Poshyvanyk, M. L. Collard,“Blending Conceptual and Evolutionary Couplings to Support Change Impact Analysis in Source Code”, Proc. *WCRE*, 2010, pp. 119 – 128.

[11] Mann-Whitney-Wilcoxon Test: http://en.wikipedia.org/wiki/Mann%E2%80%93Whitney_U_test

[12] Mann-Whitney-Wilcoxon Test Online: http://www.socscistatistics.com/tests/mannwhitney/Default2.aspx

[13] M. Mondal, C. K. Roy, K. A. Schneider, “An Empirical Study on Change Recommendation ”, Proc. *CASCON*, 2015, 10pp. (to appear

[14] M. Mondal, C. K. Roy, K. A. Schneider. “Prediction and Ranking of Co-change Candidates for Clones”, Proc. *MSR* 2014, pp. 32 – 41 .

[15] M. Mondal, C. K. Roy, K. A. Schneider, “Automatic Identification of Important Clones for Refactoring and Tracking”, Proc. *SCAM*, 2014, pp. 11 – 20.

[16] M. Mondal, C. K. Roy, K. A. Schneider, “Automatic Ranking of Clones for Refactoring through Mining Association Rules”, Proc. *CSMR-WCRE*, 2014, pp. 114 – 123.

[17] A. T. Nguyen, T. T. Nguyen, H.A. Nguyen, A. Tamrawi, H. V. Nguyen, J. Al-Kofahi, T. N. Nguyen, “Graph-based pattern-oriented, context sensitive source code completion”, Proc. *ICSE*, 2012, pp. 69 – 79.

[18] H. A. Nguyen, T. T. Nguyen, G. Wilson Jr, A. T. Nguyen, M. Kim, T. N. Nguyen, “A graph-based approach to API usage adaptation”, *ACM Sigplan Notices*, 45(10): 302 – 321.

[19] T. T. Nguyen, H. A. Nguyen, N. H. Pham, J. Al-Kofahi, T. N. Nguyen, “Recurring bug fixes in object-oriented programs”, Proc. *ICSE*, 2010, pp. 315 – 324.

[20] H. A. Nguyen, A. T. Nguyen, T. T. Nguyen, T. N. Nguyen, H. Rajan, “A Study of Repetitiveness of Code Changes in Software Evolution”, Proc. *ASE*, 2013, pp. 180 – 190.

[21] Nonparametric Tests: http://sphweb.bumc.bu.edu/otlt/MPH-Modules/BS/BS704_Nonparametric/mobile_pages/BS704_Nonparametric4.html

[22] T. Omori, H. Kuwabara, K. Maruyama, “A study on repetitiveness of code completion operations”, Proc. *ICSM*, 2012, pp.584 – 587.

[23] D. M. Pletcher, D. Hou, “BCC: Enhancing code completion for better API usability”, Proc. *ICSM*, 2009, pp. 393 – 394.

[24] B. Ray, M. Nagappan, C. Bird, N. Nagappan, T. Zimmermann, “The Uniqueness of Changes: Characteristics and Applications”, *Microsoft Research Technical Report*, 2014, pp. 1 – 10.

[25] C. K. Roy, J. R. Cordy, R. Koschke, “Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach”, *Science of Computer Programming*, 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] C. K. Roy, J. R. Cordy, “Scenario-based Comparison of Clone Detection Techniques”, Proc. *ICPC*, 2008, pp.153 – 162.

[28] R. Robbes, M. Lanza, “How Program History Can Improve Code Completion”, Proc. *ASE*, 2008, pp. 317 – 326.

[29] J. Svajlenko, C. K. Roy, “Evaluating Modern Clone Detection Tools”, Proc. *ICSME*, 2014, pp. 321 – 330.

[30] M. Toomim, A. Begel, S. L. Graham, “Managing Duplicated Code with Linked Editing”, Proc. *VL/HCC*, 2004, pp. 173 – 180.

[31] N. Tsantalis, A. Chatzigeorgiou, “Ranking Refactoring Suggestions Based on Historical Volatility”, Proc. *CSMR*, 2011, pp. 25 – 34.

[32] T. Wang, M. Harman, Y. Jia, J. Krinke, “Searching for better configurations: a rigorous approach to clone evaluation”, Proc. *ESEC/FSE*, 2013, pp. 455 – 465.

[33] C. Zhang, J. Yang, Y. Zhang, J. Fan, X. Zhang, J. Zhao, P. Ou, “Automatic parameter recommendation for practical API usage”, Proc. *ICSE*, 2012, pp. 826 – 836.

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