Micro-clones in Evolving Software

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

***Abstract*—Detection, tracking, and refactoring of code clones (i.e., identical or nearly similar code fragments in the code-base of a software system) have been extensively investigated by a great many studies. Code clones have often been considered bad smells. While clone refactoring is important for removing code clones from the code-base, clone tracking is important for consistently updating code clones that are not suitable for refactoring. In this research we investigate the importance of micro-clones (i.e., code clones of less than five lines of code) in consistent updating of the code-base. While the existing clone detectors and trackers have ignored micro clones, our investigation on thousands of commits from six subject systems imply that around 80% of all consistent updates during system evolution occur in micro clones. The percentage of consistent updates occurring in micro clones is significantly higher than that in regular clones according to our statistical significance tests. Also, the consistent updates occurring in micro-clones can be up to 23% of all updates during the whole period of evolution. According to our manual analysis, around 83% of the consistent updates in micro-clones are non-trivial. As micro-clones also require consistent updates like the regular clones, tracking or refactoring micro-clones can help us considerably minimize effort for consistently updating such clones. Thus, micro-clones should also be taken into proper consideration when making clone management decisions.**

***Index Terms*—Code Clones; Micro-Clones; Consistent Updates; Software Evolution;**

I. Introduction

Code cloning is a common yet controversial practice which is often employed by the programmers for repeating similar functionalities during software development and maintenance [41], [44], [43]. Code cloning refers to the task of copying one code fragment from one place of a code-base and pasting it to several other places with or without modifications. Such copy/paste activities result the existence of identical or nearly similar code fragments, known as code clones, at different places of the code-base. A group of similar code fragments is known as a clone class. Beside copy/paste activities there can be several other reasons behind creating code clones [45]. Whatever may be the reasons behind code clones, these are of significant importance from the perspectives of software maintenance and evolution.

A great many studies [2], [3], [4], [11], [12], [13], [16], [20], [23], [24], [25], [28], [29], [40], [33], [34], [55], [26], [27], [51], [49], [15], [47], [58], [39] have been done on detecting and analyzing code clones. While a number of studies [2], [12], [13], [20], [23], [24], [25] discover some positive impacts (such as faster software development, reduction of development costs and efforts) of code clones on software development and maintenance, other studies [3], [16], [28], [11], [29], [33], [34], [26], [51], [15] have identified negative impacts (such as hidden bug-propagation, unintentional inconsistencies, late propagation, and high instability) with strong empirical evidence. Emphasizing the issues related to code clones researchers suggest to manage them through refactoring and tracking.

Motivation. Clone size is an important parameter while detecting code clones for analysis and management. The default value of this parameter is 50 tokens for token-based clone detectors such as: PMD [38], iClones [10], and CCFinder [19]. This value for line-based clone detectors (such as: NiCad [7], ConQAT [17], Simian [50]) is 6 lines. According to the study of Wang et al. [57], the best value of this parameter (i.e., lower threshold of clone size) for NiCad, ConQAT, and Simian is 5 lines for detecting clones in Java and C source code. For PMD, iClones, and CCFinder, this best value is 26 tokens. While most of the studies have used the default value of this clone size parameter, code clones of minimum 6 lines have been considered in the benchmarking experiments [52], [5]. Modern clone detectors are moving towards a larger minimum clone size of 10 to 15 lines or similar [48], [53]. Detecting code clones with a particular minimum threshold of clone size is important. Code clones with a size which is smaller than the minimum threshold (such as single or double line clones) might not appear to be important for management. However, from our manual investigation of the changes that occurred to the source code of our subject systems we experience that code clones with a very small size, such as single line or two line clones which are not part of other bigger clones, might often be important for software maintenance such as consistently updating the code-base. Focusing on this finding, in this study we introduce the concept of micro-clones and investigate their importance in software maintenance and evolution. We define micro-clones in the following way.

Micro-Clones: *Code clones having a size that is smaller than the minimum size of regular code clones are called micro-clones in our study. Micro-clones are not parts of regular code clones.* The minimum size of a micro-clone fragment is 1 LOC. In our experiment we use the NiCad clone detector for detecting regular clones. We detect regular code clones of at least 5 lines which is the best threshold value for NiCad clone detector for detecting code clones from Java and C source code as was reported by Wang et al. [57]. Thus, in our experiment, a micro-clone fragment can have at most 4 LOC. The existing studies have considered code clones of less than 5 LOC to be false positives, and have ignored them. Clone detection researchers report that code clones of less than 6 LOC are meaningless [5]. To the best of our knowledge, our study is the first one to investigate the importance of micro-clones in software evolution and maintenance.

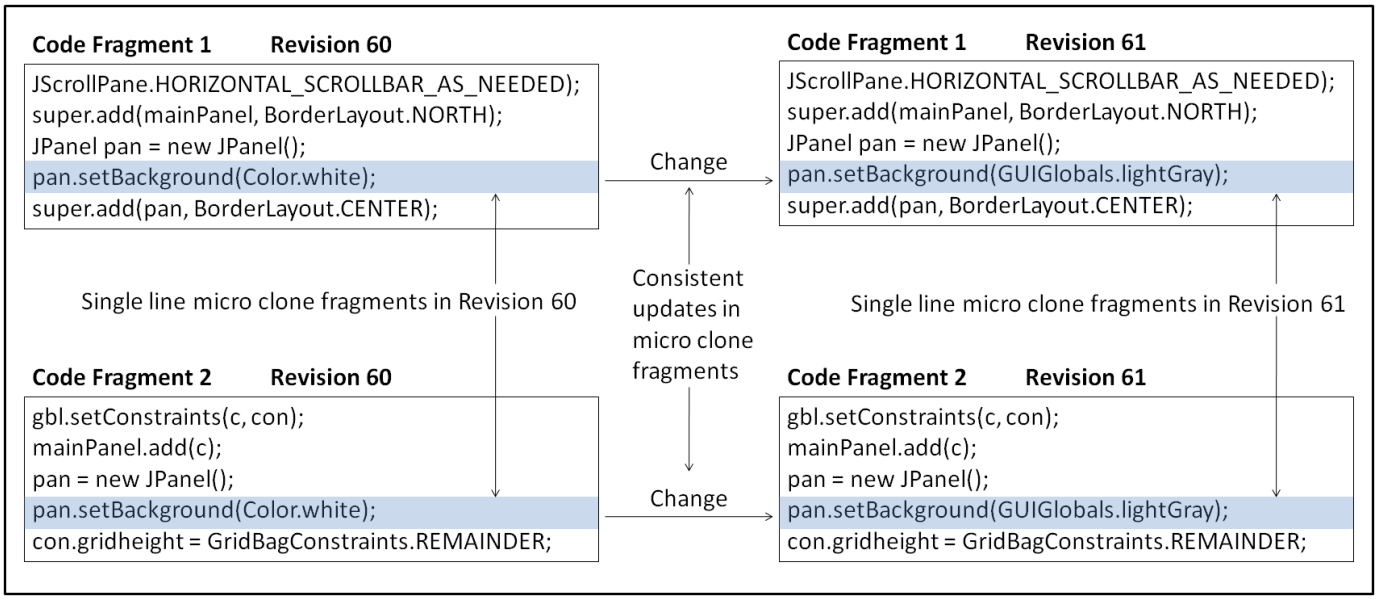

Fig. 1. The figure shows consistent updates (modifications) in a pair of single line micro clones from our subject system Jabref. We can see two code fragments (Code Fragment 1 and Code Fragment 2) in revision 60. We also highlight the single line micro clones residing in these two fragments in revision 60. The corresponding code fragments along with the micro clones in revision 61 have also been shown in the figure. We see that the micro clone fragments were updated (changed) consistently in the commit operation which was applied on revision 60. From the figure we also realize that the surrounding code of one micro clone fragment is not similar to that of the other micro clone fragment.

TABLE I
RESEARCH QUESTIONS

SL	Research Question
RQ 1	What percentage of the consistent changes occur in micro clones?
RQ 2	Are the consistent changes occurring in micro-clones non-trivial?
RQ 3	What proportion of the code clones in a software system are micro clones?
RQ 4	What type of consistent updates (addition, modifications, or deletions) are more frequent in micro-clones?
RQ 5	What is the size of the micro-clones that experience most of the consistent updates?
RQ 6	Do the micro clone fragments that experience consistent updates remain in the same file or in different files?

Fig. 1 shows examples of single line micro-clones from our subject system Jabref. The figure shows that the micro clone fragments were updated consistently in the commit operation which was applied on revision 60. Figure caption contains the details. Fig. 1 implies that consistent changes can also occur in micro clones. Thus, such clones should not be ignored when making clone management decisions. Automatic support for tracking (or refactoring when possible) micro-clones can help us minimize consistent update effort for such clones.

Approach. We perform our investigation on thousands of revisions of six open-source software systems written in Java and C. Considering each system we detect regular code clones of at least 5 LOC (the best minimum threshold for NiCad) from each of its revisions using the NiCad clone detector. We also identify the changes that occurred to the code-base in each of the commit operations. We identify those changes that are related to consistent updates of the code-base. We then determine which of these consistent updates occurred to the regular clones detected by NiCad and which ones occurred to the micro-clones. We analyze these consistent updates and answer six research questions listed in Table I.

Findings. According to our analysis on thousands of commits of six subject systems written in Java, and C:

- Overall around 80% of all consistent updates during the whole period of evolution occur in micro clones. The regular code clones experience only 16% of the consistent updates. Section IV-A discusses the remaining 4% of the consistent updates. The percentage of consistent updates (consistent additions, deletions, and modifications) occurring in micro clones is significantly higher than that in regular clones. The consistent updates that occur in micro clones can be up to 23% of all updates (additions, deletion, and modifications) during system evolution.
- Around 83% of the consistent updates occurring in micro-clones are non-trivial (i.e., the updates have effects on program execution and output).

The current studies on clone management have only considered regular clones ignoring the micro-clones. However, our findings imply that micro clones should also be considered equally important for management. Our research also reveals that the number of micro-clones is much higher than the number of regular clones in a software system. We also analyze different characteristics of micro-clones. The findings from our analysis are important from the perspectives of micro-clone management.

Paper Organization. The rest of our paper is organized as follows. Section II describes the terminology, Section III discusses the experimental steps, Section IV presents our experimental results and analyzes those results to answer the research questions, Section V discusses the related work,

Section VI mentions the possible threats to validity, Section VII discusses the reproducibility of our research, and Section VIII concludes the paper by mentioning future work.

II. Terminology

A. Different Types of Code Clones

Our research involves detection and analysis of code clones of all three major clone-types: Type 1, Type 2, and Type 3. We define these clone-types in the following way according to the literature [42], [41].

- **Type 1 Clones.** The identical code fragments residing in a software system's code-base are called Type 1 clones. More elaborately, if two or more code fragments in a code-base are exactly the same disregarding their comments and indentations, then we call these code fragments identical clones or Type 1 clones of one another.
- **Type 2 Clones.** Syntactically similar code fragments residing in a software system's code-base are known as Type 2 clones. Type 2 clones are generally created from Type 1 clones because of renaming identifiers and/or changing data types.
- **Type 3 Clones.** Type 3 clones, also known as near-miss clones, are generally created from Type 1 or Type 2 clones because of additions, deletions, or modifications of lines in these clones.

B. Consistent Updates in Code Clones

Code clones have a tendency of being updated consistently. Clone fragments from the same clone group might often experience the same or similar changes during evolution. We define consistent updates in code clones in the following way.

Consistent Update: *Let us consider a clone-pair (two code fragments that are similar to each other) in a particular revision of a subject system. A commit operation was applied on the revision, and both of the clone fragments were changed in the commit. If the two clone fragments experienced the same or similar change in the commit operation, then we say that the clone fragments were updated consistently.*

In our research we consider the same change case. Thus, if each of the two clone fragments in a clone-pair experience the same change in a commit operation we consider that the clone fragments were updated consistently in the commit.

C. Identifying Consistent Updates in Code Clones

Let us consider that the two clone fragments from a clone-pair were changed in a particular commit operation. We determine these changes using UNIX diff. Diff outputs three types of changes: addition, modification, and deletion. We consider that the clone fragments were updated consistently if the following conditions hold:

- In case of addition, the same line(s) were added after the same line in each clone fragment.
- In case of modification, the same line(s) in each clone fragment should be modified in the same way. In other words, the lines that were modified and the lines that we obtain after modification should be the same in each clone fragment.
- In case of deletion, the same line(s) should be deleted from each clone fragment.

We can check these conditions by comparing the diff outputs corresponding to the changes in the two clone fragments.

TABLE II
Subject Systems

Systems	Lang.	Domains	LLR	Revs
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
Ctags	C	Code Def. Generator	33,270	774
Camellia	C	Image Processing Library	89,063	170

LLR = LOC in the Last Revision Revs = No. of Revisions

III. Experimental Steps

We conduct our experiment by downloading six subject systems written in Java and C from an on-line SVN repository [37]. Our subject systems have been listed in Table II. We select these systems for our study, because these are of diverse variety in terms of application domains, and size. The revision histories of these systems are also of different lengths. Moreover, the systems are written in two different programming languages. We intentionally select our subject systems emphasizing their diversity so that we can generalize our findings. We perform a number of experimental steps for each of the systems. Fig. 2 shows the sequential flow of these steps. A brief description of each of these steps is given below.

- Downloading each of the revisions (as mentioned in Table II) of the subject system from the SVN repository.
- Preprocessing the source code files in each revision by removing comments and blank lines.
- Detecting changes between the corresponding source code files of every two consecutive revisions by applying UNIX diff.
- Detecting code clones from each of the revisions by applying the NiCad clone detector.
- Mapping the changes that occurred to each revision to the already detected code clones in that revision by using line numbers of the changes and clones.
- Identifying consistent updates by following the procedure described in Section II.
- Identifying consistent updates in regular clones and micro-clones following the procedure in Section IV-A.

We detect code clones using the well known clone detector NiCad [7] that can detect all three types of clones (Type 1, Type 2, and Type 3) with high precision and recall [44], [46]. A recent study [54] shows that NiCad is a good choice among the modern clone detectors in term of detection accuracy. As suggested in Wang et al.'s [57] study, we detect regular code clones of at least 5 LOC using NiCad. We also use NiCad for detecting micro-clones of at most 4 LOC.

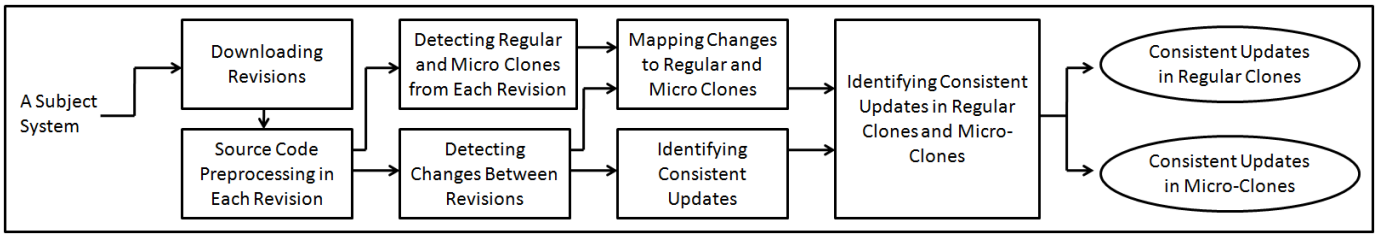

Fig. 2. Experimental steps for detecting consistent updates in regular clones and micro-clones

IV. Experimental Results and Analysis

In this section, we present our experiments, report our experimental results, and analyze our results to answer the research questions in Table I.

A. Answering the First Research Question (RQ 1)

RQ 1: *What percentage of the consistent updates (i.e., consistent changes) occur in micro clones?*

Motivation. Answering this question is important. As the regular code clones require consistent updates, the existing clone trackers consider regular clones for tracking so that such clones can be updated consistently with reduced effort. If we find that micro clones also require consistent updates, then micro clones should also be tracked by the clone trackers. Tracking micro-clones will help us minimize effort for consistently updating such clones. We perform out investigation in the following way.

Methodology. We analyze each of the commit operations of a subject system for answering RQ 1. Considering a particular commit we extract the changes (using UNIX diff) that occurred in the code-base. Following the procedure described in Section II we identify the consistent updates. We first determine which of these consistent updates occurred in regular code clones. The remaining consistent updates occurred in micro-clones. In the following paragraph we provide a clarifying example.

Let us consider a pair of consistent updates. In each of these updates, a single line of code was modified (e.g., Fig. 1). Let us consider that in one update the line l_1 was changed to $l_{1,modified}$. In the other update, the line l_2 was changed to $l_{2,modified}$. These two updates occurred in the same commit operation. As these updates are consistent updates, we can realize that the two lines l_1 and l_2 are identical, and also, the lines $l_{1,modified}$ and $l_{2,modified}$ are identical. We first examine whether the two lines l_1 and l_2 (i.e., the lines before the occurrence of the updates) belong to the regular code clones detected by NiCad. If this is true, then we consider these updates as consistent updates to regular clones. However, if the identical lines l_1 and l_2 do not belong to regular clones, then they are parts of a micro clone pair. Let us consider that the micro clone fragments mcf_1 and mcf_2 make this pair where mcf_1 contains l_1 and mcf_2 contains l_2. These micro clone fragments can be of 4 LOC most. They can even be single line micro clones, and in this case, mcf_1 and mcf_2 only consist of the lines l_1 and l_2 respectively. The surrounding code of l_1 is not similar to that of l_2 in this case.

We detect all three types of regular code clones using the NiCad clone detector, and then determine which of the consistent updates occurred in these regular clones. The remaining consistent updates occurred in micro-clones. We should note that two consistent updates that did not occur in regular clones were considered to have occurred in micro-clones if these updates satisfy the following conditions:

- In the case of consistent additions, none of the additions was made after a line that contains only a single character.
- In the case of consistent deletions, none of the deletions involved deleting only a single line that contains only a single character.
- In the case of consistent modifications, none of the modifications was made to a single line that contains only a single character.

We apply these conditions, because we were careful that lines containing a single character should not be considered as micro-clones. However, we did not apply these conditions when determining whether consistent updates occurred in regular clones, because regular clones might have lines containing only a single character and additions, deletions or modifications might happen to such lines. Such consistent updates can also occur to micro-clone fragments that contain more than one line. However, our restrictions regarding micro-clones help us in avoiding unmeaningful micro-clones. By examining all the commit operations of a subject system we determine the following measures:

- **AU (All Updates)**: The total number of updates (additions, deletions, and modifications) that occurred during the whole period of system evolution.
- **ACU (All Consistent Updates)**: The total number of consistent updates.
- **CURC (Consistent Updates in Regular Clones)**: The total number of consistent updates in regular clones.
- **CUMC (Consistent Updates in Micro Clones)**: The total number of consistent updates in micro-clones.
- **UCU (Uncategorized Consistent Updates)**: The total number of consistent updates that neither occurred in regular clones nor were considered to have occurred in micro-clones. Such consistent updates involve additions after, deletions of, or modifications to lines containing only a single character.

In Table III, we show these measures for each of our subject systems. We see that the number of uncategorized consistent updates (UCU) is very low for each system. For three subject

TABLE III
NUMBER OF CONSISTENT UPDATES IN REGULAR AND MICRO CLONES

Systems	AU	ACU	CURC	CUMC	UCU
jEdit	6261	669	0	554	115
Freecol	16320	3565	500	3063	2
Carol	8253	1714	0	1551	163
Jabref	14917	1777	861	904	12
Ctags	2114	109	35	74	0
Camellia	3118	749	0	719	30

AU = Total number of updates during the whole period of evolution.
ACU = Total number of consistent updates during whole evolution period.
CURC = Total number of consistent updates in regular code clones.
CUMC = Total number of consistent updates in Micro-clones.
UCU = Total number of consistent updates that neither occurred in regular clones nor were considered to have occurred in micro-clones.

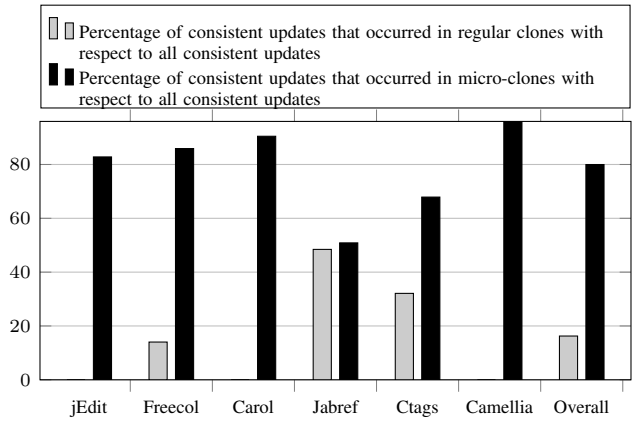

Fig. 3. Comparing the percentages of consistent updates in regular clones and micro-clones with respect to all consistent updates

systems, jEdit, Carol, and Camellia, there were no consistent updates in regular clones. For each of the candidate systems, the number of consistent updates in micro clones is higher compared to regular clones.

We also determine the percentages of consistent updates that occurred in regular clones and micro-clones with respect to all consistent updates. These percentages for each of our candidate systems are shown in Fig. 3. From the figure it is clear that the percentage regarding micro clones is always greater than the percentage regarding regular clones. We also show the overall percentages (the two right most bars in Fig. 3) considering all subject systems. We see that while overall 80% of all consistent updates occur in micro clones, only 16% of the consistent changes occur in regular clones. The remaining 4% of the consistent updates are uncategorized consistent updates.

Statistical Significance Test. We also wanted to determine whether the percentage of consistent updates in micro clones is significantly higher than the percentage of consistent updates in regular clones. We investigate six subject systems in total. Thus, we have six percentages for micro clones, and six corresponding percentages for regular clones. We perform Mann-Whitney-Wilcoxon (MWW) tests [30], [31]

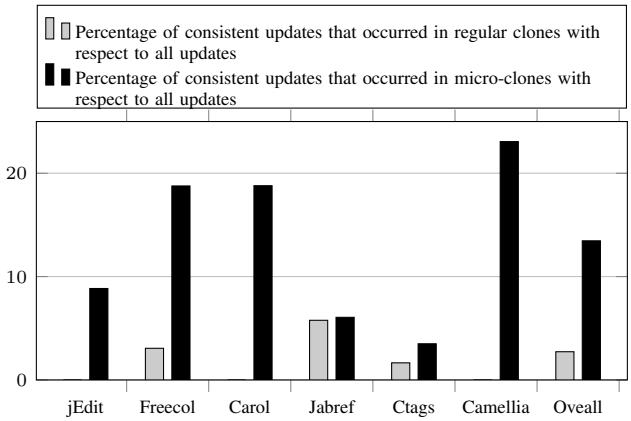

Fig. 4. Comparing the percentages of consistent updates in regular clones and micro-clones with respect to all updates

to determine whether the six percentages for micro clones are significantly different than those for regular clones. We conduct our tests considering a significant level of 5%. We should note that MWW test is non-parametric [30], and thus, it does not require the samples to be normally distributed. MWW test can be applied to both small and large sample sizes. From our tests we see that the percentages regarding micro clones are significantly different than the percentages regarding regular clones with a *p-value* of 0.00512 for the two-tailed test case. As the percentage regarding micro clones is always higher than the percentage regarding regular clones, we realize that *the percentage of consistent updates in micro clones is significantly higher than the percentage of consistent updates in regular clones*.

Fig. 4 shows the percentage of consistent updates in micro clones and regular clones with respect to all updates during the whole period of evolution. From the graph we realize that the percentage of consistent updates that occur in micro clones can often be considerable with respect to all updates during evolution. The graph shows that overall around 13.46% of all updates are consistent updates that occur in micro-clones. This percentage for regular clones is only 2.73%. The percentage regarding micro clones of our subject system Camellia is the highest one (around 23%) according to Fig. 4.

Answer to RQ 1: From our investigation and analysis we can state that around 80% of all consistent updates during system evolution occur in micro clones. Also, only 16% of the consistent updates occur in regular clones. According to our statistical significance tests, the percentage of consistent updates occurring in micro clones is significantly higher compared to regular clones. We also find that the consistent updates that occurred in micro clones can be up to 23% of all updates.

Our findings imply that we should not ignore micro clones while making clone management (refactoring and/or tracking) decisions, because micro clones experience a significantly higher percentage of consistent updates compared to regular

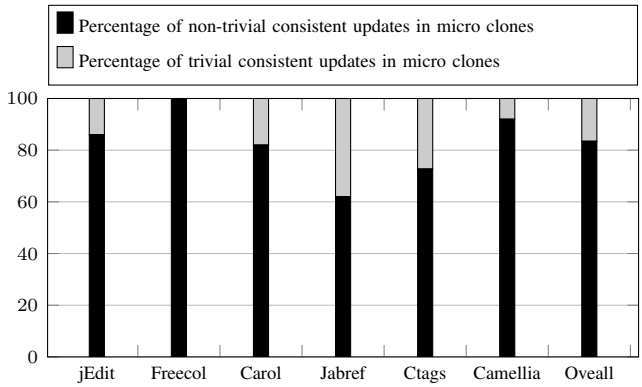

Fig. 5. Comparing the percentages of trivial and non-trivial consistent updates in micro-clones

clones. Tracking or refactoring (when possible) micro-clones can help us minimize effort for consistent updates in such clones. The existing studies on detection and management of code clones only focus on regular clones. From our findings we believe that micro clones should also be considered equally important for refactoring and tracking. Possibly, tracking is the most suitable technique for managing micro clones, because these are very small in size compared to regular clones. In a previous study on regular clones, Zibran et al. [60] found that larger regular clones are more attractive to the programmers for refactoring. As micro clones are even smaller than the minimum size of the regular clones, micro clones may not be promising to the programmers for refactoring. In the next research question (RQ 2) we analyze whether the consistent updates occurring in micro-clones are trivial or not.

B. Answering the Second Research Question (RQ 2)

RQ 2: *Are the consistent changes occurring in micro-clones non-trivial?*

Motivation. In order to realize the importance of micro-clones, we need to know whether the consistent updates occurring in micro-clones are at all meaningful. If we see that micro-clones experience non-trivial consistent updates during evolution, then it should be a good motivation behind considering micro-clones for management. We manually analyze the consistent changes that occurred in micro-clones to determine whether they are trivial or not. We perform our investigation in the following way.

Methodology. We identify the consistent updates (additions, modifications, and deletions) in micro clones by sequentially examining the commit operations of a subject system as described in RQ 1. We record the distinct updates in an HTML file, and then manually analyze these updates to determine whether they are trivial or not. We previously noted that we process the source code in each revision by removing comments and blank lines before applying UNIX diff and identifying consistent updates. Kawrykow and Robillard [21] investigated non-essential changes in the code-base. They report that rename-induced modifications, local variable extraction, trivial keyword modification, and whitespace updates can be considered as non-essential changes. We mark such changes as trivial ones during our manual analysis of the HTML files. From each of the subject systems we record the first 50 distinct consistent updates in micro-clones during our sequential examination of the commit operations. From our six candidate systems we record 300 consistent updates in total. We manually analyze these updates and annotate each one as a trivial or non-trivial one. We then determine the percentage of trivial and non-trivial consistent updates in micro-clones. We show these percentages for each subject system in Fig. 5. The figure shows that the percentage of non-trivial consistent updates is much higher than the corresponding percentage of trivial consistent updates in each subject system. All the consistent updates in micro clones of our subject system Freecol are non-trivial. We see that overall around 83% of the consistent updates can be non-trivial. Such an observation complies with the findings of Kawrykow and Robillard [21]. They found that overall 15% of the updates in a system can be trivial updates. In our manual investigation this percentage is 17% (=100-83) overall.

For each of the subject systems we generate a HTML file that contains the first 50 distinct consistent updates in micro-clones of that system. We annotate the HTML files by marking each consistent update as a trivial or a non-trivial one. These HTML files are available on-line [6]. During our manual analysis we observe that the non-trivial consistent updates mostly occur in method calls, assignment statements, and if-conditions residing in micro-clone fragments.

> **Answer to RQ 2:** According to our analysis, most of the consistent updates (83% of the consistent updates) occurring in micro-clones are non-trivial updates.

Our findings imply that the consistent changes occurring in micro-clones are mostly non-trivial. Thus, micro-clones should not be ignored when deciding about clone management.

C. Answering the Third Research Question (RQ 3)

RQ 3: *What proportion of the code clones in a software system are micro clones?*

Motivation. From our answer to RQ 1 we realize that most of the consistent updates during system evolution occur in micro clones rather than regular clones. We wanted to make a comparison between amounts of micro clones and regular clones in a software system. If we see that the number of micro clones is much higher compared to the number of regular code clones in a software system, then it is possibly more important to investigate the micro clones and see which ones are more likely to experience consistent updates. We answer RQ 3 in the following way.

Methodology. For answering RQ 3 we investigate the last revision (the latest revision as mentioned in Table II) of each of our subject systems. We first apply the NiCad clone detector on the latest revision in order to detect the regular clones of at least 5 lines of code. Then, we again apply NiCad for detecting code clones with a minimum of one line and a maximum of 4 lines of code. However, many of these code clones can reside

TABLE IV
NICAD SETTINGS FOR DETECTING REGULAR AND MICRO-CLONES

	Clone Granularity	Min Line	Max Line	Identifier Renaming	Dissimilarity Threshold
Regular Clones	block	5	20000	blind renaming	20%
Micro Clones	block	1	4	blind renaming	20%

TABLE V
NUMBER OF DISTINCT REGULAR AND MICRO CLONE FRAGMENTS IN EACH OF OUR SUBJECT SYSTEMS

	jEdit	Freecol	Carol	Jabref	Ctags	Camellia
CRC	3823	305	226	171	139	93
CMC	6146	4449	2244	2306	284	203

CRC = Count of Regular Clones. CMC = Count of Micro Clones.

in the already detected regular clones. By excluding such code clones that reside in regular clones we get the micro clones.

Table V shows the number of distinct regular and micro clone fragments in the last revision of each of our candidate systems. We detected regular clones and micro clones of all three types (Type 1, Type 2, and Type 3) using the settings in Table IV. Section II contains the definitions of these three clone-types. By looking at the settings we can realize that there can be no Type 3 micro clones with such settings. A micro clone fragment can only have 4 lines of code at most. Let us consider a micro clone pair that consists of micro clone fragments each having 4 lines of code. If one line in a clone fragment differs than the corresponding line of code in the other fragment, then the proportion of dissimilarity between the two fragments becomes 25% (one line is different within 4 lines), and it exceeds the dissimilarity threshold as mentioned in Table IV. Thus, the number of micro-clones reported in Table V include Type 1 and Type 2 micro clones only.

From Table V we realize that the number of micro clones in a subject system is generally higher than the number of regular clones. Considering all subject systems we find that the number of micro clones is around three times higher than the number of regular clones. In Fig. 6 we can also see the comparison of the amounts of regular and micro clones in percentages. The overall percentages of regular and micro clones considering all subject systems are around 23.33% and 76.67% respectively.

Answer to RQ 3: From our experimental results and analysis we realize that the number of micro clones in a software system is generally higher than the number of regular clones in that system. Micro clones are around thrice in quantity compared to regular clones.

For each subject system we create two HTML files. One file contains distinct pairs of Type 1 micro clones, and the other file contains distinct pairs of Type 2 micro clones. We previously mentioned that Type 3 micro clones are not possible with the settings (Table IV) that we have used. These two files for each subject system are available on-line [6]. By looking at the Type

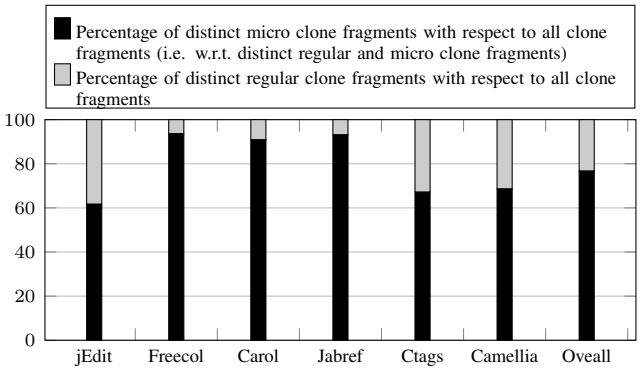

Fig. 6. Comparing the percentages of regular and micro-clones

1 and Type 2 micro clone pairs it seems that Type 1 micro clones are more suitable for management (such as tracking or refactoring) than Type 2 ones. However, the number of Type 2 micro clones is much higher than Type 1 micro clones. We also observe that micro clones mostly consist of small methods, if-blocks, else-blocks, try-blocks, and catch-blocks.

From RQ 1 and RQ 3 we realize that the number of micro clones in a software system is generally much higher compared to regular clones, and more importantly, most of the consistent changes during software evolution occur in the micro clones. Moreover, our answer to RQ 2 implies that the consistent updates occurring in micro-clones are mostly non-trivial. As the number of micro clones is very high in a software system, it is possibly important to analyze which ones have higher possibilities of experiencing consistent updates, and which type of consistent updates they mostly experience. These analyses can help us devise mechanisms for efficient management of micro clones. We perform such analyses in the following research questions.

D. Answering the Fourth Research Question (RQ 4)

RQ 4: *What type of consistent updates (addition, modifications, or deletions) are more frequent in micro-clones?*

Motivation. Answering RQ 4 is important from the perspective of clone tracking. If it is observed that a particular type of consistent update (additions, deletions, or modifications) is more likely to occur in micro-clones compared to other types, then this information can help us develop a change type sensitive clone tracker. When a clone tracker having the capability of tracking micro-clones will sense that particular type of update in a micro clone fragment, it can assume a high likeliness that the other micro clone fragments in the same group (i.e., in the same micro clone class) will also require that update to keep the micro clone fragments consistent. We perform our investigation in the following way.

Methodology. By following the methodology described in RQ 1 we identify the consistent updates that occurred in micro clones. Considering each of these updates we determine whether this is an addition, modification, or deletion. We examine all the consistent updates that occurred in micro-clones during the whole period of evolution and determine how many of these are consistent additions, modifications, or

TABLE VI
NUMBER OF DIFFERENT TYPES OF CONSISTENT UPDATES IN MICRO CLONES

Systems	CUMC	CAMC	CMMC	CDMC
jEdit	554	64	387	103
Freecol	3063	282	2592	189
Carol	1551	298	1085	168
Jabref	904	120	710	74
Ctags	74	18	42	14
Camellia	719	43	649	27

CUMC = Total number of consistent updates in Micro-clones.
CAMC = Total number of consistent additions in Micro-clones
CMMC = Total number of consistent modifications in Micro-clones
CDMC = Total number of consistent deletions in Micro-clones

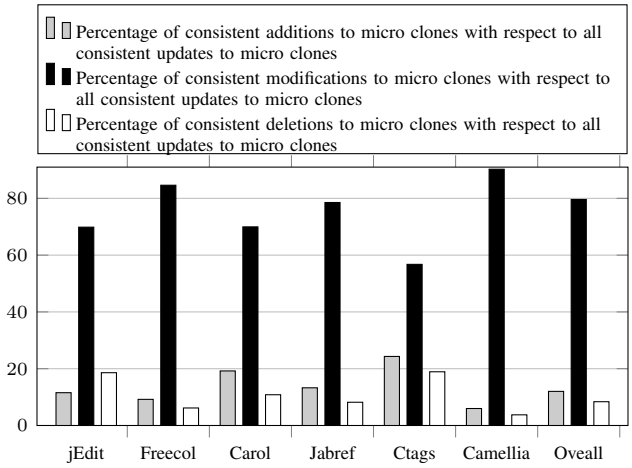

Fig. 7. Comparing the percentages of different types of consistent updates in micro-clones

deletions. We show these numbers in Table VI for each of our candidate systems. From the table we see that the number of consistent modifications in micro clones is generally much higher than the number of consistent additions, and consistent deletions. Fig. 7 shows the percentages of different types of consistent updates with respect to all consistent updates during the whole evolution. The graph clearly demonstrates that the percentage of consistent modifications is always higher than the corresponding percentages of additions or deletions. We also show the overall percentages considering all subject systems. We see that overall around 80% of all consistent updates in micro clones are consistent modifications. The overall percentages for consistent additions and deletions are 12% and 8% respectively.

We also wanted to understand whether the percentage of consistent modifications in micro clones is significantly higher compared to the percentage of consistent additions and deletions. We performed Mann-Whitney-Wilcoxon tests [30], [36], [31] for this purpose. We first determine whether the percentages of consistent modifications from our six subject systems are significantly different than the six percentages regarding consistent additions. We perform our tests considering a significance level of 5%. From our tests we realize that the percentages of consistent modifications are significantly different than those of consistent additions with a *p-value* of 0.005 for the two tailed test case. As the percentage of modifications is always higher, we can decide that *the percentage of consistent modifications to micro clones is significantly higher than the percentage of consistent additions to such clones*. We also perform MWW tests [30], [36], [31] in a similar way to determine whether the percentages of consistent modifications are significantly higher than the percentage of consistent deletions. Our test results (with a *p-value* of 0.005) again confirm that *the percentage of consistent modifications is significantly higher than the percentage of consistent deletions*.

Answer to RQ 4: According to our investigations and analysis, most of the consistent updates that occur in micro clones are consistent modifications. We also find that the proportion of consistent modifications to micro clones is significantly higher than the proportions of consistent additions or deletions to such clones.

Our findings imply that a micro clone tracker should primarily focus on notifying programmers when a micro clone fragment will experience a modification so that the programmer does not miss to propagate the same modification to the other micro clones in the same group if necessary.

E. Answering the Fifth Research Question (RQ 5)

RQ 5: *What is the size of the micro-clones that experience most of the consistent updates?*

Motivation. Answering RQ 5 is important for developing a micro clone tracker. If it is observed that most of the consistent updates occur in the micro clones of a particular size (in term of lines of code), then we can mainly consider detecting and tracking micro-clones of that particular size. We perform our investigation in the following way.

Methodology. We first identify the consistent updates that occurred in micro-clones by following the procedure described in RQ 1. Considering each pair of consistent updates that occurred in a micro-clone pair, we determine the size of the micro clone fragments in the pair. Determining the size of the micro clone fragments is tricky. We do it in the following way.

Let us consider two consistent modifications occurred to two lines l_1 and l_2 in the code-base. The lines l_1 and l_2 are identical according to our considerations in Section II. As we mentioned in RQ 1, l_1 and l_2 can be parts of a micro clone pair where each of the two fragments in the pair contains more than one line of code. We analyze the code surrounding l_1 and l_2 and then determine two code fragments with maximum size such that:

- One code fragment contains the line l_1, and the other fragment contains l_2.
- The code fragments exhibit Type 1 or Type 2 similarity, and thus, they are of the same size.
- The fragments are disjoint in the case they belong to the same source code file.

The above conditions ensure that the two code fragments make a micro clone pair. The surrounding code (i.e., the

TABLE VII
NUMBER OF CONSISTENTLY UPDATED MICRO CLONE PAIRS OF DIFFERENT SIZES

	jEdit	Freecol	Carol	Jabref	Ctags	Camellia
Size 1	237	566	1053	182	31	950
size 2	121	12866	265	138	24	940
Size 3	107	22754	504	88	4	176
Size 4	113	698	1024	125	7	377

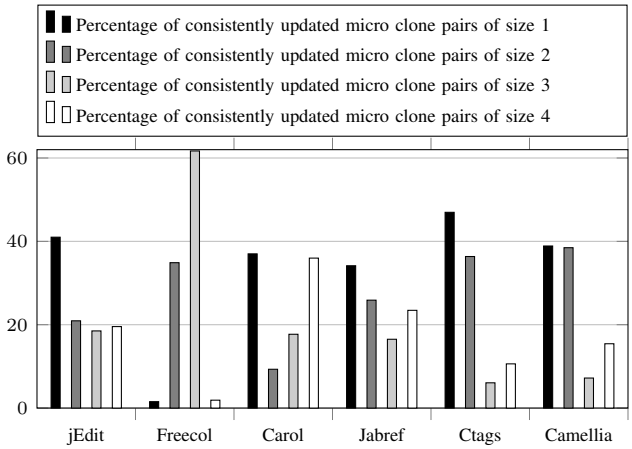

Fig. 8. Percentages of consistently updated micro clone pairs

contexts) of these two micro clone fragments are different, because we determine the biggest fragments that are similar to each other. We record the size of the clone fragments in the pair. Size is the number of source code lines in a fragment of the clone pair. We can easily understand that the lowest possible value of this size is one, and the highest possible value is four. Table VII shows the number of consistently updated micro clone pairs with different size for each subject system. Fig. 8 shows the percentages of such micro clone pairs of different sizes. From Fig. 8 we realize that the percentage of consistently updated micro clone pairs of size one is the highest for four subject systems: jEdit, Carol, Jabref, and Ctags. In case of Freecol, the percentage regarding size three is the highest. The graph implies that the percentage of consistently updated single line micro clone pairs is the highest for most of the subject systems.

Answer to RQ 5: According to our investigation, *single line micro clones experience the highest percentage of the consistent updates for most of our candidate systems. The percentage regarding double line micro clones is also very high for most of the cases.*

Our finding implies that it is better not to ignore micro clones of any size from consideration. However, single line micro clones have the highest possibility of experiencing consistent updates for most of the subject systems.

F. Answering the Sixth Research Question (RQ 6)

RQ 6: *Do the micro clone fragments that experience consistent updates remain in the same file or in different files?*

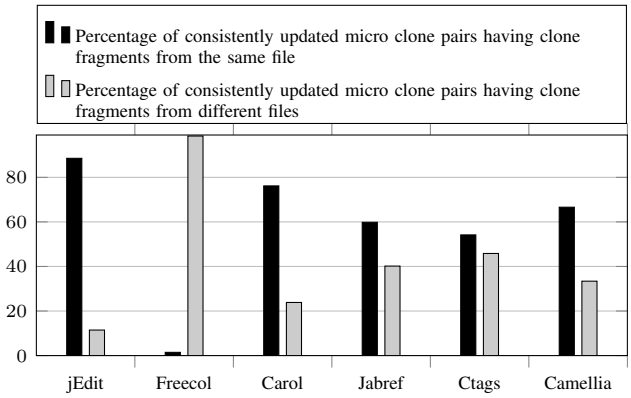

Fig. 9. Percentages of consistently updated micro clone pairs having clone fragments from the same file or different files

Motivation. Answering this question is important for detecting and tracking micro-clones. If it is observed that the micro clone fragments that experience consistent updates generally reside in the same source code file, then we can primarily focus on detecting and tracking micro clones from the same file. Thus, the answer from RQ 6 might help us in efficient designing of a micro-clone detector and tracker. We investigate this in the following way.

Methodology. We identify the consistent updates in micro clones by following the procedure described in RQ 1. We analyze each pair of consistent updates that occurred in micro clones, and determine whether the micro clone fragments experiencing the consistent updates reside in the same file or in different files. We perform this analysis for all pairs of consistent updates that occurred in micro clones during the whole period of evolution, and then determine the number of cases where the consistent updates in a pair occurred in the same file or in different files. In Fig. 9, we show the percentages of such cases with respect to all pairs of consistent updates. The figure shows that the percentage of consistently updated micro clone pairs consisting of clone fragments from the same file is most of the time (for five out of six subject systems) higher than the percentage of consistently updated pairs consisting of clone fragments from different files.

Answer to RQ 6: According to our experimental results, *micro clone fragments that experience consistent updates mostly remain in the same file.* However, the subject systems: Freecol, Jabref, and Ctags in Fig. 9 show that *micro clone pairs each having clone fragments from different files can often require consistent updates. For our subject system Freecol, the percentage of micro clone pairs each having clone fragments from different files is much higher compared to the percentage of micro clone pairs each having clone fragments from the same file.*

V. RELATED WORK

A great many studies [2], [3], [11], [12], [13], [16], [20], [23], [24], [25], [22], [28], [29], [40], [33], [34], [55], [26], [51], [15], [47], [58], [59] have been done on detecting,

analyzing, and managing code clones. Software researchers suggest that code clones having a size of less than 6 lines of code are meaningless from the perspective of management [5]. None of the existing studies have detected and analyzed code clones of less than 5 lines of code for refactoring and/or tracking considering that these are possibly spurious clones in terms of maintenance. However, in our study we investigate the importance of micro clones (code clones of less than 5 LOC) in software evolution.

A number of studies [8], [9], [14], [56], [32] have investigated clone tracking. The main purpose of clone tracking is to ensure consistent updates to the code clones. The existing studies only consider regular clones for consistent updating. From our study we find that 80% of the consistent updates occur in micro-clones during evolution. The regular code clones only experience 16% of the consistent updates. Our study thus implies that micro-clones should also be considered for tracking.

A number of studies [16], [20], [23], [24], [25], [28], [29], [33], [34] have investigated the impacts of code clones on software evolution and maintenance. However, these studies were conducted considering the regular code clones only. In our study we investigate the importance of micro-clones in software evolution. Micro-clones have been ignored by the existing studies on clone impact.

A number of studies [18], [1], [61] have been conducted on detecting and analyzing the evolutionary coupling of software entities such as files, classes, and methods. The goal of these studies is to identify which entities have a tendency of getting changed together. If two or more entities have a tendency of changing together, then while changing one of these entities in future we can suggest the other entities to the programmer to make changes to these entities consistently with that particular entity. Most of the studies on evolutionary coupling have been done on file level evolutionary coupling. Such studies identify which other files might need to be changed consistently when a programmer attempts to change a particular file. There are some studies on method level evolutionary coupling too. In our study we discover that micro-clones have a tendency of being updated consistently. Thus, our study contributes to discovering evolutionary coupling considering a finer granularity (code fragment level granularity of even 1 line).

From our previous discussion it is clear that the existing studies on clone analysis have ignored micro-clones. Our experimental results imply that micro-clones should be considered equally important when making clone management decisions. The findings from our research questions are important towards building a clone tracker considering micro-clones. We believe that our research makes a significant contribution towards better management of software systems.

VI. Threats to Validity

We conduct our investigations by detecting regular code clones using the NiCad clone detector [7]. For different settings of the clone detector, the experimental results and findings can be different. Wang et al. [57] mentioned this problem as the *confounding configuration choice problem* and performed an in-depth investigation in order to find the most suitable configurations for different clone detectors. The setting that we have used for NiCad for detecting regular code clones was suggested to be the most suitable one by Wang et al. [57]. Thus, we believe that our findings are important for consistent evolution of software systems.

In our experiment we did not study enough subject systems to be able to generalize our findings regarding the importance of micro clones. However, our candidate systems were of diverse variety in terms of application domains, sizes, revisions, and implementation languages. Thus, we believe that our findings cannot be attributed to a chance, and these are important from the perspectives of clone management.

VII. Verifiability of Research

The clone detection tool (NiCad [7]) as well as the subject systems that we have used in our experiment are available on-line [35], [37]. The NiCad settings that we have used for detecting regular and micro clones have been shown in Table IV. While answering the research questions we have presented tables containing the raw data obtained from our investigations. We have created HTML files for our manual analysis, and these files are also available on-line [6]. Thus, our research presented in this paper is reproducible.

VIII. Conclusion

In our research, we investigate the importance of micro clones (code clones of 4 LOC or less) in software maintenance and evolution. The existing studies have always ignored such small clones considering that those are spurious clones. However, from our experiments on thousands of revisions of six diverse subject systems written in Java and C languages we realize that around 80% of the consistent updates during the whole period of evolution occur in micro clones. The percentage of consistent updates occurring in micro clones is significantly higher than the percentage of consistent updates in regular clones (code clones of 5 LOC or more). We also find that the consistent updates that occur in micro clones can be around 23% of all updates during system evolution. Thus, micro clones should not be ignored when making clone management decisions. Automatic support for tracking micro clones can help us considerably minimize effort for making consistent updates to such clones. We manually analyze the consistent updates occurring in micro-clones, and realize that most of these updates (around 83%) are non-trivial. From our further analysis we discover that consistent modifications are the dominant consistent updates in micro clones. For most of our subject systems, single line micro-clones have a higher tendency of experiencing consistent updates compared to micro-clones of other sizes. We believe that our findings are important for developing a micro-clone tracker. We plan to develop such a clone tracker in the future considering our findings from this research. The HTML files regarding our manual analysis of the consistent updates in micro-clones are available on-line [6].

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] L. Barbour, F. Khomh, Y. Zou, "An empirical study of faults in late propagation clone genealogies", *Journal of Software: Evolution and Process*, 2013, 25(11):1139 – 1165.

[5] S. Bellon, R. Koschke, G. Antoniol, J. Krinke, and E. Merlo. Comparison and evaluation of clone detection tools. IEEE Trans. Softw. Eng., 33(9), 2007.

[6] Consistent updates in micro clones. http://goo.gl/EOjeRY

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

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

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

[10] N. Göde, R. Koschke, "Incremental clone detection", Proc. *CSMR*, 2009, pp. 219 – 228.

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

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

[13] K. Hotta, Y. Sano, Y. Higo, S. Kusumoto, "Is Duplicate Code More Frequently Modified than Non-duplicate Code in Software Evolution?: An Empirical Study on Open Source Software", Proc. *EVOL/IWPSE*, 2010, pp. 73 – 82.

[14] P. Jablonski, 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, pp. 16 - 20.

[15] L. Jiang, Z. Su, E. Chiu, "Context-Based Detection of Clone-Related Bugs", Proc. *ESEC-FSE*, 2007, pp. 55 – 64.

[16] E. Juergens, F. Deissenboeck, B. Hummel, S. Wagner, "Do Code Clones Matter?", Proc. *ICSE*, 2009, pp. 485 – 495.

[17] E. Juergens, F. Deissenboeck, and B. Hummel, "Clonedetective - a workbench for clone detection research", Proc. *ICSE*, 2009, pp. 603 – 606.

[18] 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.

[19] T. Kamiya, S. Kusumoto, and K. Inoue, "Ccfinder: a multilinguistic token-based code clone detection system for large scale source code" *IEEE Trans. on Softw. Eng.*, 2002, 28(7):654 – 670.

[20] C. Kapser, M. W. Godfrey, ""Cloning considered harmful" considered harmful: patterns of cloning in software", *Empirical Software Engineering*, 2008, 13(6): 645 – 692.

[21] D. Kawrykow, M. P. Robillard, "Non-Essential Changes in Version Histories", Proc. *ICSE*, 2011, pp. 351 – 360.

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

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

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

[25] J. Krinke, "Is Cloned Code older than Non-Cloned Code?", Proc. *IWSC*, 2011, pp. 28 – 33 .

[26] J. Li, M. D. Ernst, "CBCD: Cloned Buggy Code Detector", Proc. *ICSE*, 2012, pp. 310 – 320.

[27] Z. Li, S. Lu, S. Myagmar, Y. Zhou, "CP-Miner: A Tool for Finding Copy-paste and Related Bugs in Operating System Code", Proc. *OSDI*, 2004, pp. 20 – 20.

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

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

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

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

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

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

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

[35] NiCad Clone Detector: http://www.txl.ca/nicaddownload.html

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

[37] On-line SVN repository: http://sourceforge.net/

[38] PMD: http://pmd.sourceforge.net/

[39] F. Rahman, C. Bird, P. Devanbu, "Clones: What is that Smell?", Proc. *MSR*, 2010, pp. 72 – 81.

[40] D. C. Rajapakse and S. Jarzabek, "Using Server Pages to Unify Clones in Web Applications: A Trade-off Analysis", Proc. *ICSE*, 2007, pp. 116 – 126.

[41] C. K. Roy, M. F. Zibran, R. Koschke, "The Vision of Software Clone Management: Past, Present, and Future (Keynote paper)", Proc. *CSMR-WCRE*, 2014, pp. 18 – 33.

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

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

[44] 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.

[45] C. K. Roy, J. R. Cordy, "A Survey on Software Clone Detection Research", Tech Report TR 2007-541, School of Computing, Queens University, Canada, 2007.

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

[47] R. K. Saha, C. K. Roy, K. A. Schneider, "An Automatic Framework for Extracting and Classifying Near-Miss Clone Genealogies", Proc. *ICSM*, 2011, pp.293 – 302.

[48] H. Sajnani, V. Saini, J. Svajlenko, C. K. Roy, and C. V. Lopes, "Sourcerercc: Scaling code clone detection to big code", Proc. *ICSE*, 2016, pp. 1157 – 1168.

[49] G. M. K. Selim, L. Barbour, W. Shang, B. Adams, A. E. Hassan, Y. Zou, "Studying the Impact of Clones on Software Defects", Proc. *WCRE*, 2010, pp. 13 - 21.

[50] Simian: http://www.harukizaemon.com/simian/

[51] D. Steidl, N. Göde, "Feature-Based Detection of Bugs in Clones", Proc. *IWSC*, 2013, pp. 76 – 82.

[52] J. Svajlenko, C. K. Roy, "Evaluating clone detection tools with bigclonebench", Proc. *ICSME*, 2015, pp. 131 – 140.

[53] J. Svajlenko, I. Keivanloo, C. K. Roy,"Big data clone detection using classical detectors: an exploratory study", *Journal of Software: Evolution and Process*, 2015, 27(6):430 – 464.

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

[55] S. Thummalapenta, L. Cerulo, L. Aversano, M. D. Penta, "An empirical study on the maintenance of source code clones", *Empirical Software Engineering*, 2009, 15(1): 1 – 34.

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

[57] 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.

[58] X. Wang, Y. Dang, L. Zhang, D. Zhang, E. Lan, H. Mei, "Can I clone this piece of code here", Proc. *ASE*, 2012, pp. 170 – 179.

[59] S. Xie, F. Khomh, Y. Zou, "An Empirical Study of the Fault-Proneness of Clone Mutation and Clone Migration", Proc. *MSR*, 2013, pp. 149 – 158.

[60] M. F. Zibran, R. K. Saha, C. K. Roy, "Genealogical Insights into the Facts and Fictions of Clone Removal", 2013, *ACR*, 13(4): 30 – 42.

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