Understanding the Evolution of Type-3 Clones: An Exploratory Study

Ripon K. Saha* Chanchal K. Roy† Kevin A. Schneider† Dewayne E. Perry*
*The University of Texas at Austin, USA
†University of Saskatchewan, Canada
ripon@utexas.edu, chanchal.roy@usask.ca, kevin.schneider@usask.ca, perry@mail.utexas.edu

***Abstract*—Understanding the evolution of clones is important for both understanding the maintenance implications of clones and for building a robust clone management system. To this end, researchers have already conducted a number of studies to analyze the evolution of clones, mostly focusing on Type-1 and Type-2 clones. However, although there are a significant number of Type-3 clones in software systems, we know a little how they actually evolve. In this paper, we perform an exploratory study on the evolution of Type-1, Type-2, and Type-3 clones in six open source software systems written in two different programming languages and compare the result with a previous study to better understand the evolution of Type-3 clones. Our results show that although Type-3 clones are more likely to change inconsistently, the absolute number of consistently changed Type-3 clone classes is greater than that of Type-1 and Type-2. Type-3 clone classes also have a lifespan similar to that of Type-1 and Type-2 clones. In addition, a considerable number of Type-1 and Type-2 clones convert into Type-3 clones during evolution. Therefore, it is important to manage type-3 clones properly to limit their negative impact. However, various automated clone management techniques such as notifying developers about clone changes or linked editing should be chosen carefully due to the inconsistent nature of Type-3 clones.**

***Index Terms*—Type-3 clones; clone genealogy; clone evolution**

I. Introduction

The investigation and analysis of code clones has attracted considerable attention from the software engineering research community in recent years. Researchers have presented evidence that code clones have both positive [14], [25] and negative [18] consequences for maintenance activities and thus, in general, code clones are neither good nor bad. It is also not possible or practical to eliminate certain clone classes from a software system to minimize their potential threats [14]. Consequently, the identification and management of software clones, and the evaluation of their impact has become an essential part of software maintenance. Knowing the evolution of clones throughout a system's history is important for properly comprehending and managing its clones [13].

There are mainly three types of code clones defined by the researchers based on textual similarity–Type-1, Type-2, and Type-3. Type-1 clones are identical code fragments but may have some variations in whitespace, layout and comments. They are also known as identical clones. Type-2 clones are syntactically equivalent fragments with some variations in identifiers, literals, types, whitespace, layout and comments. Type-3 clones could include all the changes of Type-1/Type-2 clones with further modifications such as changed, added or removed statements. The dissimilarity threshold of a clone detector determines how much dissimilar fragments could be in the same clone class. In some research papers, Type-2 and Type-3 clones both are referred together as near-miss clones.

There has been quite a bit of research on studying the evolution of code clones. These studies retrospectively investigated how clone classes are modified through versions for different levels of granularity, for different subject systems written in different languages, using different clone detection tools, and from different perspectives. However, most of the studies [2], [14], [15], [23], [25] focused on only Type-1 and Type-2 clones. Therefore, the existing knowledge regarding the evolution of Type-1 and Type-2 clones is reasonably rich. However, we know only a little about the evolution of Type-3 clones despite the fact that there are substantially more Type-3 clones than Type-1 or Type-2 [22].

Recently, Bazrafshan [5] studied seven subject systems at revision level to analyze the evolution of near-miss clones and compared with that of identical clones. Based on their findings Bazrafshan found that near-miss clones seem to be more dominating and should get more attention in maintenance. Although this is a good study towards understanding the evolution of near-miss clones, there are two important reasons that further analysis is necessary to understand the evolution of Type-3 clones properly. First, Bazrafshan analyzed the evolution of Type-2 and Type-3 clones together as near-miss clones. Therefore, it is hard to understand the unique behavior of Type-3 clones. Second, researchers found widely varying results in their studies of the evolution of Type-1 and Type-2 clones depending on the subject systems, the tools, and the granularity of analysis. We expect the same to be true of studying the evolution of Type-3 clones.

In this paper, we conduct an exploratory study on the evolution of Type-3 clones to understand their maintenance implications in more detail. Unlike the previous study, we separated all the three types of clones (instead of considering Type-2 and Type-3 clones together as near-miss clones), and analyzed their evolution independently for different subject systems at a different level of granularity (release level rather than revision level), using a different clone detection tool and genealogy extractor. Then we compared the results of Type-3 clones with that of Type-1 and Type-2 clones to understand the unique behavior of Type-3 clones. More specifically we answer the following research questions.

RQ1: What is the lifetime of Type-3 clone classes compared to Type-1 and Type-2?

Lifetime is an important metric to understand the importance of managing a particular type of clones. If a clone class disappears very quickly, it would not be cost effective to manage/refactor it. In previous studies [6], [23], researchers found many Type-1/Type-2 clones that are long lived, and thus need to be managed. In this study, we want to see if the same results hold for Type-3 clones as well.

RQ2: Do Type-3 clone classes change more frequently than Type-1 and Type-2?

Change frequency is another important metric to assess the benefits of a clone management system. The more a clone class changes, the more additional costs it may incur, since changing one clone fragment typically requires the changes to be propagated to all of its siblings. Furthermore, the likelihood of an unintentional inconsistent change increases as the number of changes increases.

RQ3: Do Type-3 clone classes exhibit similar change patterns to those of Type-1 and Type-2?

Understanding the change patterns of Type-3 clones is very important to design appropriate techniques for managing them. For example, if Type-3 clones mostly tend to change consistently, tool support such as linked editing would be useful to prevent unintentional inconsistent changes.

RQ4: Do Type-1 or Type-2 clone classes become Type-3 clone classes during evolution or vice versa?

In order to design a robust clone management system, the type of a clone class could be an important parameter to select appropriate techniques. During the course of evolution, it is possible that one type of clone class converts to another type. For example, a Type-1/Type-2 clone class could be converted into Type-3 in the next version due to an inconsistent change or vice versa. Knowing what percentage of clone classes convert into different types is important in this regard.

RQ5: What are the frequent syntactic changes to Type-3 clone classes during evolution? Do those changes also happened in Type-1 and Type-2? Is there any syntactic change that happens more consistently than others?

While numeric data regarding clone evolution are useful for understanding the high level behavior of code clones, understanding the characteristics of how clones actually evolve may open new approaches to clone management [14]. As a first step, we investigate the extent to which syntactic changes are consistent or inconsistent w.r.t. the previous version. Given a history of consistent changes, a useful strategy is to notify the developers that an inconsistent changes might be unintentional.

To summarize the result of this research, we have observed that Type-3 clones changed 28% more frequently than Type-1 and Type-2 clones, and 70% of the changed Type-3 genealogies were inconsistent. Furthermore, in 16% of Type-3 clone classes, changes took place in the dissimilar lines. In the study period, we have found a considerable number of Type-1 and Type-2 clones that converted into Type-3 clones due to inconsistent changes. But interestingly, the absolute number of consistently changed Type-3 clone classes was higher than that of Type-1 and Type-2 clone classes. Type-3 clones also maintained a lifetime similar to that of Type-1 and Type-2 clones. Therefore, managing Type-3 clones is very important and existing clone management support such as synchronous editing would be helpful for Type-3 clones as well. But since most of Type-3 clone classes tend to be changed inconsistently, this kind of automatic support should be chosen very carefully.

Our paper makes the following contributions:

1) We extend gCad [24], a near-miss clone genealogy extractor, to get more detailed data on the evolution of Type-3 clones.
2) Our study presents empirical data on the evolution of Type-3 clones, which not only verify some previous findings in different experimental settings, but also show that the behavior of Type-3 clones is far different from Type-2 clones during evolution, although both of them are categorized as near-miss clones.
3) We investigate the relationship between change patterns of different types of clones and their syntactical changes. Our initial results suggest that the type of syntactic changes may provide useful hints about the consistency of changes.
4) Based on the study results, we suggest several approaches to deal with Type-3 clones that would be helpful to design a robust clone manage system.

The rest of this paper is organized as follows. Section II defines important terminology. Section III introduces the new gCad features. Section IV describes the study procedure. Section V presents empirical results. We discuss different maintenance implications of Type-3 clones in Section VI and threats to validity in Section VII. In Section VIII we discuss the related work, and finally Section IX concludes the paper with our directions for future research.

II. Terminology

Clone Genealogy: A clone genealogy is a directed acyclic graph that describes the evolution history of a clone class during a given epoch.

Consistent Change (*CC*): All clone fragments in the same clone class have been changed similarly, and thus all of them are again part of the same clone class in the next version.

Inconsistent Change (*IC*): All clone fragments in the same class have not been changed consistently. Here we should note that for Type-3 clones, all the clone fragments of a particular clone class could still form the same clone class in the next version even if one or more fragments of that class was changed inconsistently. The dissimilarity between the fragments of a clone class usually depends on the similarity threshold of the associated clone detection tools.

Static Genealogy *(SG)*: Static genealogy refers to those genealogies in which the clone fragments are propagated through subsequent releases without any textual change. The first genealogy in Figure 1 represents a static genealogy.

Consistently Changed Genealogy *(CCG)*: If a genealogy contains at least one consistent change pattern and does not contain any inconsistent change patterns, it will be classified as a consistently changed genealogy. The second genealogy in Figure 1 is an example of a *CCG*.

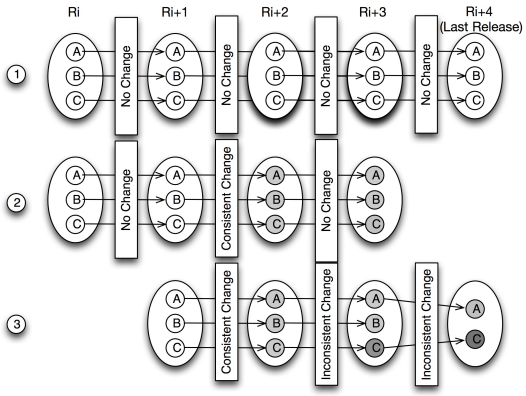

Fig. 1. Different Types of Clone Genealogies

Inconsistently Changed Genealogy *(ICG)*: If a genealogy contains any inconsistent change pattern, it will be classified as an inconsistently changed genealogy. The third genealogy in Figure 1 is an *ICG*.

Alive Genealogies *(AG)*: A genealogy is called alive genealogy if the associated clone class is still evolving and thus exist in the final version that we considered. Both the first and the last genealogies in Figure 1 are *AG*.

Dead Genealogies *(DG)*: A genealogy is called dead genealogy if its clone class disappear before reaching the final version that we considered. The second genealogy in Figure 1 represents a DG.

Lifetime of a clone class: Lifetime of a clone class represents how long the clone class exists in the associated system. In our study, we quantify lifetime in terms of number of releases. For example, if a clone class appears in the *i*th release, stays in the system until the *j*th release, we say that the lifetime of the clone class is *(j-i+1)* releases.

III. Extending gCad

Extracting the history of Type-3 clones and classifying their change patterns automatically are challenging due to the potential diverse variety of clone fragments even in the same clone class. In our previous work [24], we introduced gCad, an automatic framework for extracting and classifying near-miss clone genealogies. gCad can identify six types of clone genealogies: same, add, delete, static, consistently, and inconsistently changed genealogies. In this paper, we further extend gCad by introducing two modes of operations called *Liberal Mode* and *Strict Mode* to understand the consistent and inconsistent changes of Type-3 clones in more detail.

In the previous version of gCad, we have used a very conservative definition of inconsistent changes of Type-3 clones that if any changes to the clone fragments in the same clone class are different, no matter whether they are in the gaps or in the common parts, it will be considered as an inconsistent change. The rational behind this definition is that two textually dissimilar lines can sometimes be semantically equivalent, and thus changes in the gap also could be problematic. Furthermore, if a line is inconsistently added in the middle of a clone fragment, it can affect the semantics of the later part. On the other hand, one may argue that if a change takes place in the dissimilar part of a clone fragment, it may not truly be an inconsistent change because they were already

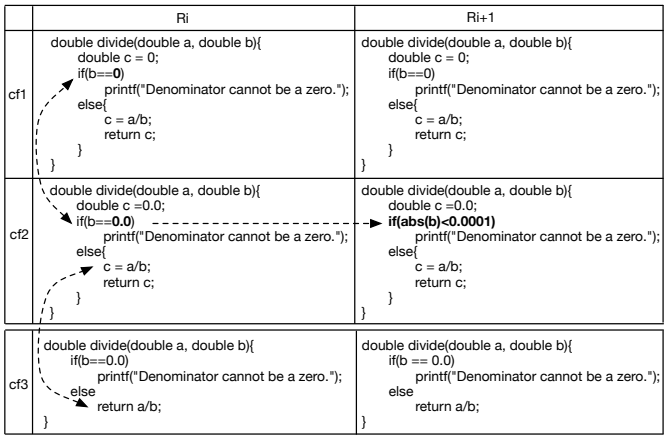

Fig. 2. An example of Type-3 clone class

dissimilar. However, each of these sides has its own advantages and disadvantages. We explain the situation with the following hypothetical example.

Let there be two clone fragments cf_1 and cf_2 in a Type-3 clone class in release R_i as shown in Figure 2 that perform a simple division operation. Although these two fragments are semantically the same, one line between these fragments is not textually the same (marked with the first dashed arrow). During the evolution of the program, a developer suddenly notices that the *if condition* of cf_2 is too precise which could result in unexpected program behavior when the value of b is very near to zero, and thus fixed the condition in release R_{i+1}. The developer did not notice that the same change is also required to be made to cf_1. Now one may think that this is a consistent change since the change took place in the gap (where source lines are not common between fragments). On the other hand, one may argue that this is an inconsistent change since one clone fragment of that clone class changed whereas another did not. We might need to see the other fragment as well to verify whether the same change is needed to that fragment or not. In the given example, the same change is also needed for cf_1.

Defining the change patterns is more complicated if there are more than two fragments in a Type-3 clone class, and the non-identical lines are not the same for all clone pairs. In order to illustrate the situation, let us assume that there is one more clone fragment (cf_3) in the example clone class shown in Figure 2. Now we see that the modified line (shown using dashed right arrow) in cf_2 is common with cf_3 but not common with cf_1. Therefore, we cannot conclude this change is a *consistent change* only considering that the change took place in the gap of clone pair (cf_1, cf_2). One of the solutions to overcome this problem is to determine the change patterns for all possible pairs in a clone class without considering the changes in the gap. If all pairs change consistently, then the change pattern will be classified as a consistent change. However, the threat explained in the previous paragraph will still be present.

Therefore, we add both approaches in gCad as two different modes of operation. The choice of mode is left to the maintenance engineer or researcher according to their context of use. The two modes are as follows:

TABLE I
SUBJECT SYSTEMS

Prog. Lang.	System	# Releases	Start Release	End Release	Start Date	End Date	Duration	LOC
Java	dnsjava	50	0.9.2	2.1.1	April 19, 1999	Feb 10, 2011	131 months	6,290-15,018
	JabRef	27	1.5	2.4.2	Aug 15, 2004	Nov 1, 2008	50 months	22,041-69,170
	ArgoUML	48	0.27.1	0.32.BETA2	Oct 4, 2008	Jan 24, 2011	26 months	176,618-202,555
C	ZABBIX	31	1.0	1.8.4	Mar 23, 2004	Jun 1, 2011	86 months	9,252-62,845
	Conky	28	1.1	1.8.1	July 20, 2005	Oct 5, 2010	62 months	6,555-39,810
	Claws Mail	44	2.0.0	3.7.9	Jan 30, 2006	April 9, 2011	63 months	1,33,642-1,89,786

1) Liberal Mode: In this mode, we ignore changes that occur in the gaps of each clone pair in the same clone class. Therefore, a change will be identified as inconsistent change only when similar lines of any clone pair in the same clone class change differently with respect to one another.

2) Strict Mode: In this mode, we do not consider the gaps a special case. If the changes to the clone fragments in the same class are not the same, it will be considered an inconsistent change.

IV. STUDY SETUP

In this section we outline our study setup for collecting relevant data to answer our research questions. It includes the choice of our subject systems, parameters for detecting clones and extracting genealogies, the overall procedure for collecting and investigating data, and a brief description of the statistical method that we used in our analysis. Since one of our main objectives is to characterize the diverse evolutionary behavior of Type-3 clones for different subject systems, clone detectors, genealogy extractors, and so on, we choose a very different settings than that of Bazrafshan's study [5].

A. Subject Systems

We studied six open source software systems for our case study. In order to select subject systems, we gave preference to those which have a long development history, and have been used in previous studies for Type-1 and Type-2 clones but not in [5]. We also ensured that multiple developers contributed to these systems so that we can assume that none is an expert of the whole system. Based on this criteria, we chose the following:

- `dnsjava`[1] is an implementation of DNS in Java.
- `JabRef`[2] is an open source bibliography reference manager that works on Windows, Linux and Mac OS X.
- `ArgoUML`[3] is an interactive, graphical software design environment that supports the design, development and documentation of object-oriented software applications.
- `ZABBIX`[4] provides a monitoring and tracking facility for network servers, devices and other resources.
- `Conky`[5] is a free, light-weight system monitor.
- `Claws Mail`[6] is an email client and news reader that support POP3, IMAP, SMTP and many other protocols.

The size of the systems (last release) varies from approximately 15K to 203K lines of code (LOC), excluding comments and blank lines. Because we want to conduct manual analysis both on the detected clones (e.g., removing false positives and uninteresting clones) and for answering the research questions, we intentionally did not choose very large systems. Of course several of the systems are reasonably large and we chose projects from different application domains to avoid biasing towards a specific kind of software system. A more detailed overview of the subject systems is presented in Table I.

[1]http://www.xbill.org/dnsjava
[2]http://jabref.sourceforge.net
[3]http://argouml.tigris.org
[4]http://www.zabbix.com
[5]http://conky.sourceforge.net
[6]http://www.claws-mail.org

TABLE II
NICAD SETTING FOR CLONE DETECTION

Setting	Value
Granularity of Clones	Block
Minimum Clone Length	5 LOC
Filtering of Statements	None
Renaming of Identifiers	Blind
UPI threshold	30%

B. Level of Granularity

In any clone evolution study, the choice of interval length between two consecutive versions to study plays a key role in the result. It is often believed that a commit/revision is not a logical unit of change. A previous study [14] also shows that there are many clones that are created for experimental purposes by the developers. However, when a version of software is officially released, the source code is expected to be in a stable form. Therefore, we should expect less inconsistent changes at release level because some consistent changes may happen in more than one commit. Therefore, we have chosen the release level instead of the revision level for our study.

C. Clone Detection

Since the main objectives of our study are to evaluate Type-3 clones, we selected NiCad-2.6.3 [8] for detecting clones. We configured NiCad with the settings given in Table II. We set the UPI threshold to 30%, which allows for 30% dissimilarity among clone fragments in a Type-3 clone class in their pretty-printed normalized format. We have chosen this threshold because this setting has been found to be very effective in detecting all the three types of clones while maintaining high precision and recall [19], [20], [22]. Furthermore, in the study of near-miss clone evolution [5], Bazrafshan concluded that different thresholds of clone detection tool influence the result but conserve the relations.

D. Genealogy Extraction

We used our *extended* near-miss clone genealogy extractor, gCad,[7] to automatically extract and classify clone genealogies for our study. In order to get more accurate results, the process of extraction and classification of genealogies is performed by gCad in two steps. In the first step, gCad classifies all the clone classes into different types (Type-1, Type-2 or Type-3), maps

[7]gCad is available online at http://www.cs.usask.ca/∼croy/

TABLE III
NUMBER OF CLONE GENEALOGIES

Systems	Type-1	Type-2	Type-3
dnsjava	6	4	132
JabRef	28	34	441
ArgoUML	162	150	1741
ZABBIX	25	33	354
Conky	9	20	162
Claws Mail	58	121	842

the clone classes and classifies their change patterns (Static or *CC* or *IC*) between each two consecutive versions. All of the mapping information and change patterns of each clone class along with their types for each consecutive versions is stored in an XML file. In the second step, gCad constructs genealogies by merging the results of each consecutive versions stored in the previous step, computes other relevant data such as frequency of changes, age of genealogies, and stores the results in another XML file. Users can also manually verify each mapping, to correct any wrong result, or to filter out any uninteresting clone genealogies. For example, if a user finds that gCad incorrectly classifies a change pattern, they can easily correct the XML file and rerun the second step of gCad to reconstruct the genealogies.

E. Procedure

We use the following procedure to obtain our results.

1) For each project, we capture all minor and major releases during the time period provided in Table I.
2) We run NiCad to detect clones in each of the releases.
3) We run gCad to extract and classify clone genealogies.
4) We review the clone genealogies manually to determine if they contain false positives. If so, we remove them from the study and update and store the changes.
5) Finally, we rerun the second step of gCad to reconstruct the genealogies with the manually corrected results.

F. Statistical Analysis

To determine if there is a significant difference in the proportions of different clone evolution patterns for different types of clones (RQ3), we use the Chi-Square test. We choose Chi-Square test because it tests differences in proportions for dichotomic data in contingency tables, with the number of rows or columns greater than two. However, this test has two limitations. It cannot estimate the *p-value* well if 1) any expected frequency is less than one, and 2) more than 20% of the frequencies have values less than five. We performed the Chi-Square test assuming a significance level of 95%.

V. RESULTS

In this section, we report our study findings with respect to the five research questions defined in the Introduction. In oder to gain a broader understanding, we answer each research question based on our study results and comparing the results with previous findings (where it is applicable). However, before going into more detail, we first present the number of genealogies in each subject system in Table III grouped by their type. From the table we see that, for each system, the number of Type-3 clone genealogies is substantially higher than that of Type-1 or Type-2. In a previous study, Bazrafshan [5] concluded that near-miss clones are more dominating than identical clones. Here we can say more specifically that Type-3 clones are more dominating than the other two types. This results again justify the importance of understanding the evolution of Type-3 clones to manage them properly.

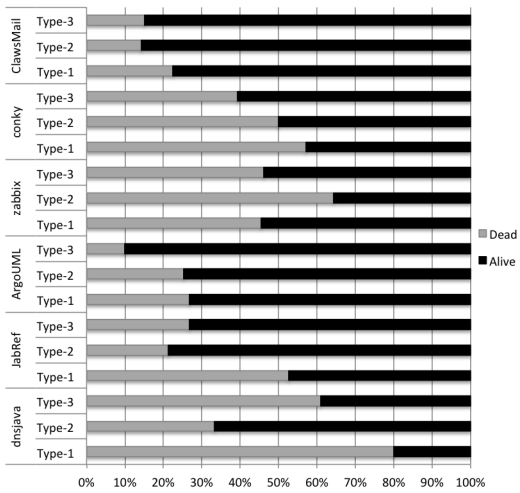

Fig. 3. Proportion of dead and alive genealogies

RQ-1: *What is the lifetime of Type-3 clone classes compared to Type-1 and Type-2?*

In order to investigate how long Type-3 clone classes exist in a system compared to that of Type-1 and Type-2, first we calculated mean lifetime of each type of genealogies. However, since the alive genealogies are still evolving and we cannot predict when they will disappear, it is not appropriate to calculate the overall mean of alive (*AG*) and dead genealogies (*DG*) together. Therefore, we calculated the mean lifetime separately for alive and dead genealogies. Figure 3 presents the proportion of *DG* and *AG* for each subject system grouped by their types. As we see from the figure, in most systems, the proportion of *AG* is higher than that of *DG* regardless of clone types. However, it is noticeable that percentages of alive Type-2 and Type-3 clones are higher than that of Type-1.

Now we take a look at Table IV, which represents the mean lifetime of *DG* and *AG* for each system grouped by clone type. By comparing the results of different types of genealogies for each system, we see that the mean lifetime of Type-2 clones for both *DG* and *AG* is slightly higher than that of Type-1 and Type-3 clones in four systems. Nonetheless, all the three types of clone classes maintain a very similar lifetime in most systems but the magnitude vary system to system. Here we also make two interesting observations regardless of clone types. First, almost for all systems, the mean lifetime of *AG* is higher than that of *DG*. Second, the lifetime of all types of *AG* for large systems (ArgoUML and Claws Mail) is substantially higher than that of smaller systems.

Comparison and Conclusion: In his study of near-miss clone evolution [5], Bazrafshan found that near-miss clones live longer than identical clones. However, we did not find any systematic relationship between lifetime and clone types. Rather, lifetime seems to be dependent on system's property (e.g. size, development practice). However, since the number of Type-3 clones is much greater than Type-1 and Type-2, the number of long lived Type-3 clones is also greater.

TABLE IV
MEAN LIFETIME OF DEAD GENEALOGIES AND ALIVE GENEALOGIES

Subject Systems	Average Lifetime of *DG*			Average Lifetime of *AG*		
	Type-1	Type-2	Type-3	Type-1	Type-2	Type-3
dnsjava	11	14	11	5	25	21
JabRef	8	5	8	15	17	15
ArgoUML	14	14	15	39	41	38
ZABBIX	9	11	10	17	15	13
Conky	8	10	9	12	6	7
Claws Mail	13	21	20	29	33	32

RQ-2: *Do Type-3 clone classes change more often than Type-1 and Type-2 clone classes?*

In previous section, we found that many long lived clones and the absolute number of such Type-3 clones is much higher than that of Type-1 and Type-2 clones. However, if they do not change, they will not incur any additional cost in maintenance. In this section, we investigate how often Type-3 clones change compared to Type-1 and Type-2.

Table V shows that Type-3 clones change more frequently than Type-1 or Type-2 clones. We observe that most Type-1 clones either do not change or only change once. We found that only two Type-1 clone classes changed in more than three releases during the evolution period. However, for each system, there is a considerable number of Type-3 clones that change in five or more releases.

One reason for a higher number of changes to Type-3 clones may be that there are more Type-3 clones in each system than the other two types. To investigate whether it is the fact, we further analyzed the data. Since most of the entries in Table V for Type-1 and Type-2 clones are less than five, we did not perform the Chi-Square test to understand the difference among types in terms of change frequency. However, we calculated the average number of changes in terms of releases for all three types of clones. The result shows that Type-3 clones change 28% more frequently than Type-1 and Type-2.

Comparison and Conclusion: Bazrafshan [5] found that near-miss clones change more frequently than identical clones. However, from our study, we can see that although Type-2 clones are part of near-miss clones, the mean number of changes to Type-1 clones and Type-2 clones is almost the same. Actually Type-3 clones change more frequently than Type-1 and Type-2 clones. One may argue that a source code fragment changes because it has to change; not because it is a Type-1, Type-2, or Type-3 clone. However, since empirically it is found that Type-3 clones changes more frequently than others (at either release or revision level), it is important to keep track of them to avoid unintentional inconsistent changes.

RQ-3: Do Type-3 clone classes exhibit similar change patterns to that of Type-1 and Type-2 clone classes?

To investigate whether the change patterns of Type-3 clones are similar to those of Type-1 or Type-2 clones, we first investigated if there exists any dependence between the change patterns of clone fragments and their types. Since changes to Type-3 clones can take place in either similar or dissimilar parts of clone fragments, we run gCad both in *strict* and *liberal* modes. In both modes, gCad automatically groups all the genealogies for each system by their associated clone types and change patterns as presented in Table VI. To illustrate,

TABLE V
CHANGE FREQUENCIES OF CLONE GENEALOGIES

Subject System	Clone Types	Number of Releases where Clones Changed						
		0	1	2	3	4	5	5+
dnsjava	Type-1	4	1	1	0	0	0	0
	Type-2	1	3	0	0	0	0	0
	Type-3	57	44	16	6	7	2	1
JabRef	Type-1	17	7	3	1	0	0	0
	Type-2	20	13	1	0	0	0	0
	Type-3	206	148	42	27	11	7	7
ArgoUML	Type-1	125	27	7	3	0	0	0
	Type-2	132	13	3	2	0	0	0
	Type-3	1239	335	101	42	20	4	3
ZABBIX	Type-1	12	8	3	2	0	0	0
	Type-2	16	10	5	1	1	0	0
	Type-3	144	125	55	14	10	6	3
Conky	Type-1	4	3	0	0	2	0	0
	Type-2	10	8	1	1	0	0	0
	Type-3	108	30	15	6	2	1	2
Claws Mail	Type-1	37	17	1	3	0	0	0
	Type-2	56	43	12	6	2	2	0
	Type-3	369	278	109	47	28	11	8

TABLE VI
NUMBER OF GENEALOGIES BY CLONE TYPES AND CHANGE PATTERNS

System	Mode	Types	Static	CC	IC	χ^2	p-value
dnsjava	-	Type-1	4	1	1		
	-	Type-2	1	1	2		
	Strict	Type-3	57	6	69	-	-
	Liberal	Type-3	57	16	59	-	-
JabRef	-	Type-1	17	2	9		
	-	Type-2	20	10	4		
	Strict	Type-3	206	43	192	21.81	0.0002
	Liberal	Type-3	206	73	162	12.38	0.01
ArgoUML	-	Type-1	125	4	33		
	-	Type-2	132	6	12		
	Strict	Type-3	1239	29	473	31.83	0.0001
	Liberal	Type-3	1239	87	415	23.78	0.0001
ZABBIX	-	Type-1	12	7	6		
	-	Type-2	16	11	6		
	Strict	Type-3	144	45	165	18.80	0.0005
	Liberal	Type-3	144	70	140	8.77	0.06
Conky	-	Type-1	4	2	3		
	-	Type-2	10	6	4		
	Strict	Type-3	108	11	43	12.90	0.01
	Liberal	Type-3	108	27	27	4.5	0.33
Claws Mail	-	Type-1	37	13	8		
	-	Type-2	56	40	25		
	Strict	Type-3	369	120	353	50.01	0.0001
	Liberal	Type-3	369	218	255	15.26	0.004

the first row of the table represents that in `dnsjava` we found four genealogies where Type-1 clone classes did not change, one genealogy where Type-1 clone class changed consistently, and one genealogy where the Type-1 clone class changed inconsistently. To investigate the relationships, we performed the Chi-Square test on a contingency table, where columns represent the genealogy change patterns (*Static*, *CC*, and *IC*) and rows represent the clone types. In this test, our null hypothesis is that *different types of clones do not have an impact on the different change patterns*. It should be noted that we could not perform the test for `dnsjava` since more than 20% of their entries have frequencies less than five. However, the χ^2 and p-values obtained from the rest of the systems for strict mode strongly suggests that there is a significant difference between the proportion of change patterns and clone types. Even when we excluded the changes in the gaps using gCad's *liberal* mode, the χ^2 and p-values suggest significant

differences (except Conky). Therefore, we can reject the null hypothesis for these subject systems.

From Table VI we see that, for most of the systems, the proportion of inconsistent changes to Type-3 clones is substantially higher than that of Type-1 and Type-2 clones when we considered the changes in the gaps (*strict* mode). On average, approximately 36% of Type-3 clones change inconsistently, whereas, this proportion is roughly 21% for Type-1 and 15% for Type-2 clones. When we investigate each system separately, we observe that all of them follow a similar trend with a small exception with `Conky`. However, it is hard to derive any conclusion from `Conky` because there were fewer Type-1 clones compared to the large number of Type-3 clones. In contrast to inconsistent changes, the proportion of consistent changes of Type-3 clones is lower than that of the other types. We found that only about 7% of genealogies are classified as *CCG* for Type-3 clones, whereas, they are 11% for Type-1 clones and 21% for Type-2 clones. From Table VI we also see a similar trend for each of the systems separately. When we change the gCad setting to *liberal* mode where we exclude changes in the gaps, the rate of inconsistent change is still higher. However the proportion of consistent changes increased significantly. We also calculate that in 16% clone classes, changes happened in gaps.

Finally, we observe that Type-3 clones are significantly less stable than Type-1 and Type-2 clones. On average, 58% of all Type-3 genealogies are static, whereas, it is 70% for Type-1 and 65% for Type-2 clone genealogies. There are two exceptions in the systems `Conky` and `ArgoUML`. In `Conky`, again it is because of the few Type-1 clones compared to Type-3 clones, whereas, in `ArgoUML` most of the clones (above 70%) were static regardless of their types.

Comparison and Conclusion: Interval between two versions has a great influence on the result of change patterns of clones. For example, every late propagation during the interval between two releases will be considered as a consistent change at release level, but will be considered as a combination of inconsistent changes at revision level. Bazrafshan [5] analyzed the change patterns at revision level whereas we analyze it at release level. Bazrafshan found that near-miss clones change more inconsistently than identical clones. If we merge our result of Type-2 and Type-3 genealogies, we also get the same finding. However, when we analyze Type-2 and Type-3 clones separately, we find that Type-2 clones are more likely to change consistently, whereas, Type-3 clones more likely to change inconsistently, although the absolute number of consistently changed Type-3 clones are greater than that of Type-1 and Type-2 clones.

RQ-4: Do Type-1 or Type-2 clone classes become Type-3 clone classes during evolution or vice versa?

From the previous sections we found that different types of clones behave differently in terms of change patterns and frequency. Therefore, clone type could be an important parameter to design robust clone management approaches. However, one type of clone classes may convert into another type during evolution. For example, a Type-1 clone class could be converted into Type-2 due to an identifier renaming in the next version, or a Type-1/Type-2 clone class could be converted into a Type- 3 in the next version due to an inconsistent change or vice versa. Since a type-sensitive clone manage management system has to cope with this situation, knowing the actual number of such conversions in systems will be helpful to make various maintenance decisions.

TABLE VII
CONVERSION OF CLONE TYPES

System	Type-1↔Type-2	Type-1↔Type-3	Type-2↔Type-3
dnsjava	0	2	4
JabRef	1	9	6
ArgoUML	3	48	47
ZABBIX	0	12	12
Conky	0	2	4
Claws Mail	2	14	41

In our study, we have found 201 changes in total, where clone classes changed their types (Table VII). Among them, most conversions are between Type-1 and Type-3 clones, and between Type-2 and Type-3 clones. However, we observe very few conversions between Type-1 and Type-2 clones. We manually investigated many of these conversions to answer *RQ-5*. We found many Type-1 and Type-2 clones became Type-3 clones, which is expected. However, we also found a considerable number of changes where Type-3 clones became Type-1 or Type-2 clones. Interestingly, many of such changes actually were late propagation, i.e., a Type-1 clone class became a Type-3 clone class by a previous change, but it again converted back to Type-1 by a late propagation. Some conversions were semantic preserving. Figure 4(a) shows a clone fragment of a Type-1 clone class in `dnsjava` that became a Type-3 clone class in release 1.1.5 only due to an omission of a *this* keyword (Figure 4(b)). Therefore, it is important to maintain this clone class as it was before the change. If a clone management tool does not support Type-3 clones, it will lose track of this type of clone classes after the conversion. This change also demonstrate the usefulness of the strict mode of gCad.

Comparison and Conclusion: Like us, Bazrafshan [5] also found some conversions between clone types during clone evolution. Since they did not provide the complete data, we cannot compare the result. Nonetheless, considering his discussion and our result, we conclude that the number of conversions is neither too large nor too small. We also have an interesting finding that whenever a Type-3 clone becomes Type-1 and Type-2, most likely it is a late propagation.

RQ-5: What are the frequent syntactic changes to Type-3 clone classes during evolution? Do those changes also happen in Type-1 and Type-2? Is there any syntactic change that happens more consistently than others?

As we noted in the Introduction, the objective of this research question is to investigate whether there are any syntactic changes that are more likely to change consistently than others and whether type is there any factor. Knowing such changes could be very helpful to find out unintentional inconsistent changes. To this end, as a first step, we manually analyzed 150 clone genealogies in our study. We wanted to investigate the same number of consistently and inconsistently changed genealogies for each type of genealogy so that we

```
public Object sendAsync(final Message query, final ResolverListener listener) {
        final Object id;
        synchronized (this) {
                id = new Integer(uniqueID++);
       }
        String name = this.getClass() + ": " + query.getQuestion().getName();
        WorkerThread.assignThread(new ResolveThread(this, query, id, listener), name);
        return id;
}
```

(a) dnsjava-**1.1.4**/org/xbill/DNS/ExtendedResolver.java

```
public Object sendAsync(final Message query, final ResolverListener listener) {
        final Object id;
        synchronized (this) {
                id = new Integer(uniqueID++);
        }
        String name = getClass() + ": " + query.getQuestion().getName();
        WorkerThread.assignThread(new ResolveThread(this, query, id, listener), name);
        return id;
}
```

(b) dnsjava-**1.1.5**/org/xbill/DNS/ExtendedResolver.java

Fig. 4. Conversion of a clone class from Type-1 to Type-3

can compare the result. Since there were only 29 consistently changed Type-1 genealogies, we selected 25 genealogies from each category. Furthermore, the sample size of 150 is sufficient (i.e, this sample size has an appropriate level of power) to detect all but the smallest effects [21].

Table VIII presents all the syntactic changes grouped by their clone type (before their change) and change pattern. We observe that although the majority of changes to Type-3 clones are either *data type* changes or line additions, for all types of clones, changes related to *if statements* occurs more frequently. Interestingly, all changes to *data types* for Type-3 clones were inconsistent, whereas most *data type* changes were consistent for Type-1 clones. No relationship was found between the changes related to *if statements* and the change pattern. However, we noticed that most line additions in Type-1 and Type-3 clones were *function calls*, whereas, they were variable assignments in Type-2 clones.

The most important finding of our manual analysis is that we found some hotspots, where most of the changes were consistent regardless of their types. They are *loop condition*, *called function name*, *variable renaming*, and *API*. For example, from Table VIII we see that there are three inconsistent changes in the *called function name* for Type-2 clones. Among them, we found two consecutive inconsistent changes in `ArgoUML` where the first inconsistent change (release 29.1→29.2) was fixed by the second inconsistent change (release 29.2→29.3). Therefore, any inconsistent changes to these hotspots could be reported as a suspicious change that may call attention to the developers to do a quick manual verification. We also noticed an interesting characteristic in the change of *called function arguments*. Whenever a parameter was added or deleted, the change was consistent. However, when there were changes in passing values, the changes were inconsistent in most cases.

Comparison and Conclusion: To the best of our knowledge, our work is the first to find any relationships between change patterns and syntactic changes. Therefore, we cannot compare our findings with any previous studies. However, based on this finding, we conclude that some syntactic changes are equally important regardless of clone type, whereas, some changes (e.g., *data type*) differ depending on the clone type. Certainly more analysis is necessary to make a comprehensive list of these type of changes. However, as a primary step, our findings again indicate that clone type and syntactic changes would be an important consideration when building an intelligent clone management system.

TABLE VIII
SYNTACTIC CHANGES TO CLONES DURING THE EVOLUTION PERIOD

Syntactic Change	Type-1		Type-2		Type-3	
	CC	IC	CC	IC	CC	IC
if statement insertion	5	3	3	7	4	3
if statement deletion	2	3	3	2	-	-
if condition change	1	-	-	5	-	1
loop condition change	-	-	-	-	2	-
called function name change	3	1	7	3	5	-
function arguments change	1	5	2	2	5	-
Line additions	3	7	6	4	3	10
Line deletions	2	2	-	-	-	-
Variable renaming	-	-	2	-	2	-
Data type change	4	1	1	1	-	9
API change	1	-	-	-	1	-
Assignment operation	1	1	1	1	3	-
Logic change	2	1	-	-	-	2
Total	25	25	25	25	25	25

VI. MAINTENANCE IMPLICATIONS

Software clone management aims at identifying and avoiding clones (preventive), organizing and managing existing clones (compensative), and removing clones (corrective) altogether to limit their negative impact. From our study we have shown that many Type-3 clones are long lived and change more frequently than others. Although Type-3 clones tend to change more inconsistently, the absolute number of consistently changed Type-3 genealogies is greater than that of Type-1 and Type-2. Therefore, compensative clone management approaches are more important for managing Type-3 clones. Two popular clone management approaches in this category are notifying developers about each change of clones instantly by tracking them [9] and linked editing. Note, however, that techniques developed for Type-1 and Type-2 clones can be also used for Type-3 clones because there are a significant number of consistently changed genealogies. But the biggest challenge of maintaining Type-3 clones is dealing with highly noisy data. Since most Type-3 genealogies tend to evolve independently, notifying developers about each change of a clone class will be annoying. Therefore, based on our findings, we believe that the following techniques would be useful to deal with the noisy data of Type-3 clones during evolution. Certainly these techniques are also applicable to Type-1 and Type-2 clones.

Machine Learning Technique: In our study, we found some general patterns such as Type-3 clones tend to change more frequently and inconsistently, all types of clones have a very similar lifespan, and so on. Although the magnitude of these metrics depend on the subject system due to different development practices, project histories, it can be learnt automatically. Furthermore, previous change patterns of clone classes and current syntactic changes could provide useful hints to predict which clones tend to change consistently or inconsistently. Machine learning techniques may prove useful (as in [27]) but that is an approach that needs to be carefully explored to determine its potential.

Incorporating Developers' Feedback: Although it may be possible to predict clone behavior using different machine learning techniques, it is not easy to detect all different types of unintentional inconsistencies among the clone fragments because it depends very much on the semantics of the code. In our study, we have found many textually dissimilar lines between two clone fragments that are semantically equivalent. In this situation, developer(s) who create or change the clone decide if the change is needed or not. Therefore, a clone management system should be able to take developer' feedback when it makes something wrong and use them appropriately to find relevant inconsistencies.

Type-sensitive management: Since different types of clones have different changing behavior, the clone type could be an important parameter to make better decisions. For example, while answering RQ4, we found many inconsistent changes that resulted a conversion from Type-1/Type-2 to Type-3 were unintentional and thus again converted back to its previous type in the next version by late propagation. Therefore, an inconsistent change to Type-1/Type-2 clones may be more interesting than Type-3 clones. Furthermore, since finding interesting Type-3 clones relevant to maintenance is more challenging than that of Type-1 and Type-2, type-specific decision would be computationally effective.

VII. Threats to Validity

A. Construct Validity

The results of a study on clones is contingent on the raw clone data, which again depends on the selection of the clone detection tool and the parameters used to detect clones. In order to mitigate this threat we chose the clone detection tool NiCad and configured the settings, which has been found to be effective detecting both exact and near-miss clones [19], [20], [22]. Furthermore, we used manual verification to ensure that there were no false positives in the result. However, NiCad may miss several clones, validation of which is out of scope of our study.

B. Internal Validity

The evolutionary data of our study vastly depends on the results of gCad. If gCad misses mapping a clone class between two consecutive versions, a single genealogy could be divided into two parts. Moreover, a clone class can wrongly map to a clone class in the next version due to an ambiguous situation. However, our experience from a previous study [24] and extensive manual analysis in this study suggest that these situations are very rare.

There might have been some unintentional errors during the manual verification due to the lack of domain knowledge or human error. However, we address this threat by discussing various ambiguous situations with other researchers in our lab who are either experts in the area or have prior experience working with code clones.

C. External Validity

We picked six open source projects as subject systems for the case study. Although we carefully chose the subject systems from different application domains, our findings may not be generalizable to other open source projects or industrial projects. This threat can be mitigated by adding more subject systems (both open source and industrial), which is part of our future work.

VIII. Related Work

Tracking clones across multiple versions of a software system and studying the evolution of clones is not a new topic. Kim et al. [14] was the first to track code clones across multiple revisions and observed how clones evolve in software systems. Based on a case study of two small Java systems, they observed that 36% to 38% of clone genealogies consist of clones that changed consistently. In another study, Cai and Kim [6] investigated the characteristics of long lived clones and predicted the survival time of clones. In our previous study [23], we extended the study of Kim et al. [14] by studying 17 open source systems of diverse variety and found no surprising results. However, in all these studies, CCFinder was used as the clone detection tool which mainly considers Type-1 and Type-2 clones. To complement these studies, in this paper, we analyze all three types of clone genealogies to understand the evolution of Type-3 clones properly.

Aversano et al. [2] performed an empirical study to investigate how clones are maintained when an evolution activity or a bug fix takes place. Based on a case study of two Java systems, they reported that either for bug fixing or for evolution purposes, most of the cloned code is consistently maintained during the same co-change or during temporally close co-changes. However, they considered only exact clones and a small number of gapped clones.

Krinke [15] analyzed five open source software systems and found that half of the changes to code clone classes are inconsistent and that corrective changes following inconsistent changes are rare. In another study [16], he found that cloned code is more stable than non-cloned code and thus concluded that cloned code requires less maintenance effort compared to non-cloned code. However, conclusions drawn from these studies were only analyzing exact clones. Later Lozano and Wermelinger [17] conducted an experiment for measuring the maintenance effort on methods consisting of both cloned code and non-cloned code. Although they found that changing a method that contains a clone may increase the maintenance effort, the characteristics analyzed in these methods did not reveal any systematic relations between cloning and increase in maintenance effort. Our study differs from these in that

instead of comparing cloned code and non-cloned code, we evaluated and compared different types of clone genealogies to answer several research questions.

Bakota et al. [3] proposed a similarity based approach for mapping clones and defined different conditions under which clones become suspicious. Battenburg et al. [4] also investigated the relationship between inconsistent changes in clones and bugs, and found that 1% to 3% of inconsistent changes introduce bugs. In this study, we focus on both consistent and inconsistent changes of clones to understand their change behavior. Furthermore, we analyzed Type-3 clones, whereas, they only analyzed Type-1 and Type-2 clones.

Duala-Ekoko et al. [9] developed a clone tracker that works in an IDE to assist developers manage clones efficiently. They tracked clones using a clone region descriptor (CRD) which is location independent. However, unlike our study, their objective was to manage Type-1 and Type-2 clones, and not to study the change patterns of Type-3 clones.

Göde [10] studied the evolution of Type-1 clones in nine open source systems, and found that the ratio of clones decreased in the majority of the systems. However, no general conclusion on the consistent or inconsistent changes to clone classes was reported. Later Göde and Koschke [11] extended the previous study by including Type-2 clones, and found that clones are changed rarely during their lifetime. If they are changed, they tend to be changed inconsistently. In another study [12], they conducted a study to assess the intentionality of the inconsistent change and reported that the rate of unintentional inconsistent changes is very small. Thummalapenta et al. [25] investigated to what extent clones are consistently propagated or independently evolved. They focused on identifying the evolution patterns of clones over time and relating those patterns with other parameters (clone granularity, clone radius and cloned code fault-proneness). On the other hand, we focused on understanding the difference of the evolution of various types of clones in terms of survival time, change patterns, change frequencies, and syntactic changes.

Bazrafshan's study [5] is the most related work to ours. We compared our results with theirs in each step where it is applicable (in Section V) to draw a more general conclusion.

IX. Conclusion

Whether clones are deemed useful or harmful, they have an impact on software maintainability. However, refactoring all the clones in a software system may not be worthwhile due to tradeoffs among the associated costs, risks, and benefits of removing clones. Thus, the recent trend in clone research focuses on the management of clones in a cost-effective way, rather than simply removing them. However, managing clones properly and in a cost-effective manner is unlikely without first understanding the diverse behavior of clones during evolution. This paper presents some concrete data on the evolution of Type-3 clones in a very different settings than previous studies and draws several broad conclusions about their change patterns, frequency, type conversions, and lifetime. Based on our findings, we show that type-3 clones should be managed more intelligently than Type-1 and Type-2 clones due to their more inconsistent nature. We have also suggested several approaches and discussed them briefly to find more relevant clones for management than uninteresting ones. We believe our insights into the evolution of Type-3 clones and recommendations to manage them would be helpful to design better clone management tools. As an immediate next step, we would like to identify different patterns of unintentional inconsistent changes by applying machine learning techniques.

References

[1] G. Antoniol, G. Casazza, M. D. Penta, and E. Merlo, "Modeling clones evolution through time series," Proc. *ICSM*, 2001, pp. 273–280.

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

[3] T. Bakota, R. Ferenc, and T. Gyimothy, "Clone smells in software evolution," Proc. *ICSM*, 2007, pp. 24–33.

[4] N. Bettenburg, W. Shang, W. Ibrahim, B. Adams, Y. Zou, and A. Hassan, "An Empirical Study on Inconsistent Changes to Code Clones at the Release Level," *SCP*, 77(6):160–176, 2012.

[5] S. Bazrafshan, "Evolution of Near-Miss Clones," Proc. *SCAM*, 2012, pp. 74–83.

[6] D. Cai and M. Kim, "An empirical study of long-lived code clones," Proc. *FASE/ETAPS*, 2011, pp. 432–446.

[7] J. R. Cordy, "The TXL source transformation language," *Sci. of Com. Prog.*, 61(3):190–210, 2006.

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

[9] E. Duala-Ekoko and M. P. Robillard, "Clone Region Descriptors: Representing and Tracking Duplication in Source Code," *TOSEM*, 20(1)3:1–31, 2010.

[10] N. Göde, "Evolution of Type-1 Clones," Proc. *SCAM*, 2009, pp. 77–86.

[11] N. Göde and R. Koschke, "Studying clone evolution using incremental clone detection," *JSME*, 2010.

[12] N. Göde and R. Koschke, "Frequency and Risks of Changes to Clones," Proc. *ICSE*, 2011, pp. 311–320.

[13] J. Harder and N. Göde, "Modeling clone evolution," Proc. *IWSC*, 2009, pp. 17–21.

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

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

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

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

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

[19] C. K. Roy and J. R. Cordy, "A mutation / injection-based automatic framework for evaluating code clone detection tools," Proc. *Mutation*, 2009, pp. 157–166.

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

[21] R. Rosenthal and R. Rosnow, *Essentials of Behavioral Research: Methods and Data Analysis*, 2nd Ed. pp. 452–453, 1991.

[22] C. K. Roy and J. R. Cordy, "Near-miss Function Clones in Open Source Software: An Empirical Study," *JSME*, 22(3):165–189, 2010.

[23] R. K. Saha, M. Asaduzzaman, M. F. Zibran, C. K. Roy, and K. A. Schneider, "Evaluating code clone genealogies at release level: An empirical study," Proc. *SCAM*, 2010, pp. 87–96.

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

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

[26] R. Tiarks, R. Koschke, and R. Falke, "An extended assessment of type-3 clones as detected by state-of-the-art tools," *SQC*, 19(2):295–331, 2011.

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

[28] M. Zibran, R. K. Saha, M. Asaduzzaman, and C. K. Roy, "Analyzing and Forecasting Near-miss Clones in Evolving Software: An Empirical Study," Proc. *ICECCS*, 2011, pp. 295–304.