Noname manuscript No.
(will be inserted by the editor)

A Comparative Study on the Intensity and Harmfulness of Late Propagation in Near-Miss Code Clones

Manishankar Mondal · Chanchal K. Roy · Kevin A. Schneider

Received: date / Accepted: date

Abstract Exact or nearly similar code fragments in a software system's source code are referred to as code clones. It is often the case that updates (i.e., changes) to a code clone will need to be propagated to its related code clones to preserve their similarity and to maintain source code consistency. When there is a delay in propagating the changes (possibly because the developer is unaware of the related cloned code) the system might behave incorrectly. A delay in propagating a change is referred to as 'late propagation' and a number of studies have investigated this phenomenon. However, these studies did not investigate the intensity of late propagation nor how late propagation differs by clone type. In this research we investigate late propagation separately for each of the three clone types (Type 1, Type 2, and Type 3). According to our experimental results on thousands of revisions of eight diverse subject systems written in two programming languages, late propagation occurs more frequently in Type 3 clones compared to the other two clone-types. More importantly, there is a higher probability that Type 3 clones will experience buggy late propagations compared to the other two clone-types. Also, we discovered that block clones are more involved in late propagation than method clones. Refactoring and tracking of SPCP clones (i.e., the clone fragments that evolve following a Similarity Preserving Change Pattern) can help us minimize the occurrences of late propagation in clones.

Manishankar Mondal
Department of Computer Science, University of Saskatchewan, Canada
E-mail: mshankar.mondal@usask.ca

Chanchal K. Roy
Department of Computer Science, University of Saskatchewan, Canada
E-mail: chanchal.roy@usask.ca

Kevin A. Schneider
Department of Computer Science, University of Saskatchewan, Canada
E-mail: kevin.schneider@usask.ca

Keywords Code Clones · Late Propagation · Clone Genealogy · Near-Miss Clones

1 Introduction

Software maintenance is one of the most important phases of the software development life cycle. Studies [GH11, GK11, LW08, LW10, Kri07, Kri08, ACP07, TCAP09, BKZ13, KG08, MRR^{+}12, MRS12c, MRS12b, MRS14c] show that code clones have both positive [Kri07, GH11, GK11, KG08] and negative [LW08, LW10, MRR^{+}12, MRS12c, MRS12b, MRS14c, ACP07, TCAP09] impacts on software maintenance and evolution. Code clones are exactly or nearly similar code fragments scattered in the code-base of a software system. These are mainly created because of the frequent copy-paste activities of the programmers with an aim to repeat the same or similar functionalities during software development and maintenance. If a code fragment is copied from one place of a code-base and pasted to some other places with or without modifications, then the original code fragment and the pasted code fragments become clones of one another.

Evolution of clones [KSNM05, ACP07, TCAP09, BKZ13] has been investigated by different studies in different ways. In this study we investigate a particular clone evolution pattern which is known as late propagation in clones according to the literature [ACP07, TCAP09, BKZ13]. There are strong empirical evidences [ACP07, TCAP09, BKZ13] that late propagation is related to bugs [ACP07, TCAP09] and inconsistencies [BKZ13] in the code-base. Researchers have investigated different specific patterns [BKZ13] of late propagation and identified which patterns are more related to bugs, faults, and inconsistencies. However, the existing studies regarding late propagation in clones have the following draw-backs.

(1) None of the studies investigate the intensities of late propagation in different types of clones separately. Such a study is important because, if late propagation is observed to be more intense in a particular clone type compared to the others, we might consider being more conscious while changing clones of that particular type. Also, we might want to refactor clones of that particular type with higher priority.

(2) None of the existing studies investigate the bug-proneness of late propagation in different types of clones separately. Such an investigation is also very important for understanding the comparative harmfulness of late propagation in different clone-types.

Focusing on these issues, we investigate late propagation in three types of clones (Type 1, Type 2, and Type 3) separately and answer seven important research questions presented in Table 1. According to our experimental results on thousands of revisions of eight diverse subject systems written in two different programming languages we can state that:

- The percentage of late propagations in Type 3 clones occurred only because of the changes in the non-matched (i.e., non-cloned) portions of the

Table 1 Research Questions

Serial	Research Question
1	What percentage of late propagations in Type 3 clones occur only because of the changes in the non-cloned portions of the participating clone fragments?
2	Are the intensities of late propagation different in different types of clones?
3	Late propagations in which types of clones are more related to bugs?
4	Clones of which clone-type(s) have higher possibilities of experiencing bug-fixing changes at the time of convergence?
5	Do the participating clone fragments in a clone pair that experience late propagation generally remain in different files?
6	Do block clones or method clones exhibit higher intensity of late propagation?
7	Do late propagations mainly occur to the SPCP clones or non-SPCP clones?

clone fragments is very low (less than one) for most of our candidate systems. However, this proportion can sometimes be considerable (for example our subject system jEdit). Such late propagations should be ignored when making clone management decisions. Our implemented system can automatically detect such ignorable late propagations. We perform our investigations related to research questions RQ 2 to RQ 7 (Table 1) by disregarding these ignorable late propagations.
– The intensity of late propagations in Type 3 clones is higher compared to the other two clone-types.
– Type 3 clones have a higher possibility of experiencing buggy late propagations compared to the clone fragments of the other two clone-types.
– Almost all of the clone fragments that experience late propagations are block clones. According to our statistical significance tests, the percentage of block clones that experience late propagations is significantly higher than the corresponding percentage of method clones. It seems that creating block clones is more risky than creating method clones.
– Around 89% of the late propagations involve SPCP clones [MRS14b] (i.e., the clone fragments that evolved following a Similarity Preserving Change Pattern called SPCP). In other words, late propagations mainly occur to the SPCP clones. By refactoring and tracking SPCP-clones we can possibly minimize future occurrences of late propagations considerably.

The rest of the paper is organized as follows. Section 2 describes the related terminology, Section 3 elaborates on the detection of late propagation in clones, the experimental results are presented and analyzed to answer the research questions in Section 4, Section 5 discusses the related work, Section 6 mentions possible threats to validity and finally, we conclude our paper by mentioning future work in Section 7. The research work presented in this paper is a significant extension of our earlier work [MRS14d]. In our previous study [MRS14d] we detected late propagations in three types of clones separately and answered three research questions (Table 1): RQ 2, RQ 5, and RQ 6. We extend this work with a number of investigations: (1) investigating which proportion of late propagations in Type 3 clones occurred only because of the changes in the non-cloned portions of the participating clone fragments, (2) analyzing and comparing the bug-proneness of the late propagations in three types of clones, and (3) investigating late propagation in SPCP clones. We perform these investigations for answering the four new research questions: RQ 1, RQ 3, RQ 4, and RQ 7.

2 Terminology

Types of Clones. We conduct our experiment regarding late propagation considering exact (Type 1) and near-miss clones (Type 2 and Type 3 clones). As is defined in the literature [Roy09], if two code fragments are exactly the same disregarding the comments and indentations, they are Type 1 clones of each other. Type 2 clones are syntactically similar code fragments. In general, Type 2 clones are created from Type 1 clones because of renaming variables or changing data types. Type 3 clones are mainly created because of additions, deletions, or modifications of lines in Type 1 or Type 2 clones.

Clone Class. A group (i.e., two or more) of clone fragments that are the same (Type 1) or similar (Type 2 or Type 3) to one another form a clone class. We detect clones using NiCad [CR11,RC08] clone detector that reports clones by grouping them into classes.

Clone Pair. Two clone fragments belonging to the same clone class form a clone pair. Thus, every possible pair (i.e., combination) of two clone fragments in a particular clone class is a clone pair. From each of the clone classes reported by NiCad [CR11] we determine all possible clone pairs for conducting our experiment.

Cloned Method. If a method contains cloned (Type 1, Type 2, or Type 3) lines, we call this method a cloned method. If all lines of a method are cloned lines, then this method is a fully cloned method.

Method Clones. If two or more methods are clones (Type 1, Type 2, or Type 3) of one another, we refer to these as method clones. Method clones are fully cloned methods.

Here, we should note that we conduct our experiment considering - (1) method clones and (2) block clones that reside in methods as was done in a previous study [BKZ13].

Late Propagation in a Clone Pair. Let us consider a pair of clone fragments. We say that this clone pair has experienced late propagation if it receives a diverging change followed by a converging change [BKZ13].

– ***Diverging Change.*** Let us assume a particular commit C_i where one or both of these two clone fragments were changed. Because of this change, the fragments were not considered as clones of each other. In other words, the clone fragments diverged. Such a change is called a *diverging change* for the clone pair.
– ***Converging Change.*** Let us assume a later commit C_{i+n} ($n >= 1$) where one or both of these fragments were changed, and because of this change,

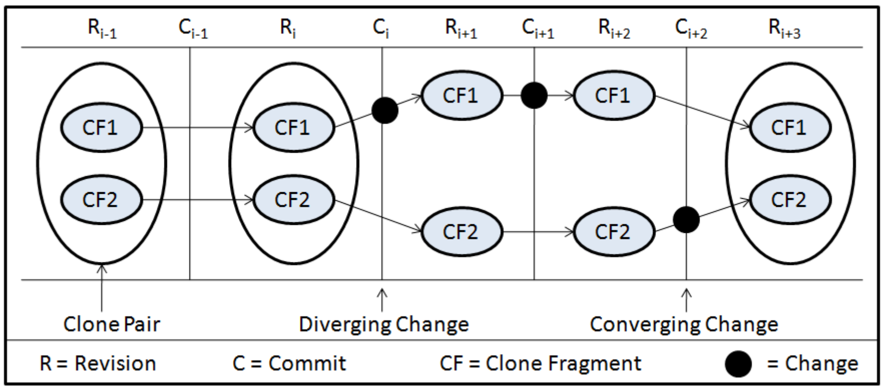


Fig. 1 A possible example of late propagation

the fragments were again considered as clones of each other. In other words, the fragments converged. The change for which the fragments converged is termed as a *converging change*.

A particular clone pair may experience late propagation more than once during evolution. Fig. 1 shows a possible example of late propagation experienced by a clone pair (*CF1*, *CF2*). The commit C_i applied on revision R_i modified *CF1* and as a result, *CF1* and *CF2* diverged. However, in commit C_{i+2}, the fragment *CF2* changed and *CF1* and *CF2* again became clones of each other in R_{i+3}.

SPCP Clones. In a previous study [MRS14b] we empirically showed that SPCP clones are important for refactoring or tracking. SPCP-Clones are those clone fragments that evolved following a particular change pattern called *Similarity Preserving Change Pattern* (SPCP). A *Similarity Preserving Change Pattern* consists of only *Similarity Preserving Change* and/or *Re-synchronizing Change*.

Similarity Preserving Change. Let us consider two code fragments that are clones of each other in a particular revision of a subject system. A commit operation was applied on this revision, and any one or both of these code fragments (i.e., clone fragments) received some changes. However, in the next revision (created because of the commit operation) if these two code fragments are again considered as clones of each other (i.e., the code fragments preserve their similarity), then we say that the code fragments received *Similarity Preserving Change* in the commit operation.

Re-synchronizing Change. Let us consider two code fragments that are clones of each other in a particular revision. If these two clone fragments experience a *diverging change* followed by a *converging change*, then we say that they experienced a re-synchronizing change. A re-synchronizing change can also be termed as a late propagation.

Here, we should clarify that two clone fragments (i.e., a clone-pair) might experience late propagation (i.e., re-synchronizing change) a number of times during evolution, however they might not be regarded as SPCP clones. Let us consider that two clone fragments experienced late propagation(s) during evolution. It might be the case that after the last occurrence of late propagation they again diverged, but did not converge again. In such a case, these two fragments will not be regarded as SPCP clones, because they did not preserve their similarity lastly. As they did not preserve their similarity, their evolution pattern is not a *Similarity Preserving Change Pattern* (i.e., is not an SPCP). Examples of such cases are evident from our answer to RQ 7.

We performed an empirical study [MRS14a] on SPCP clones where we separated all the SPCP clones in a software system into two disjoint groups. The clone fragments in one group are important for refactoring whereas, the clone fragments in the other group are important for tracking. The clone fragments that do not follow SPCP either evolve independently or are rarely changed during evolution. Thus, these non-SPCP clone fragments should not be considered important for management.

Granularity of Late Propagation. We should note that we conduct our late propagation study considering the granularity of clone pairs as was done by each of the previous studies. While it would be good to conduct such a study considering clone classes, consideration of clone classes might cause the loss of important information regarding late propagation. Let us assume a clone class consisting of six clone fragments in revision R_i. The subsequent commits might affect only two of these six clone fragments leaving the other four fragments as they are. There is a possibility that these two clone fragments (that are getting changed) will experience a late propagation (i.e., the changes occurred in one clone fragment will propagate to the other one with some delay) in future evolution. However, the other four clone fragments might not require to be changed during the whole period of evolution. In other words, the changes occurred to the two clone fragments might not ever need to be propagated to the other four clone fragments. In such a situation, consideration of all these six clone fragments for late propagation is not reasonable. While pairs of clone fragments in a particular class might experience late propagation, the whole class might not. Thus, we believe that investigating late propagation considering clone pairs is reasonable.

3 Detection of Late Propagation

We detect and experiment late propagations from eight subject systems listed in Table 2. We downloaded the revisions of each of these systems from an open-source SVN repository SourceForge[1]. For each of the subject systems we considered each of the revisions beginning from the first one. We select these systems in our research focusing on the diversity of their application domains (i.e., the systems belong to seven application domains), sizes (i.e., the subject systems are of different sizes, from very small to large), and the

[1] Sourceforge: `http://www.sourceforge.net`

Table 2 Subject Systems

Systems	Lang.	Domains	LOC	Revisions
Ctags	C	Code Def. Generator	33,270	774
Camellia	C	Image Processing Library	89,063	207
BRL-CAD	C	3D Modeling	40,941	735
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
Java-ML	Java	Java Machine Learning Library	16,428	1200

number of revisions. Thus, we believe that our reported experimental results are not affected by these parameters.

3.1 Preliminary Steps

Detection of late propagation by mining the revisions of a particular subject system requires the following preliminary steps to be done sequentially - (i) Extraction of methods from each of the revisions, (ii) Detection of method genealogies, (iii) Extraction of clones from each of the revisions, (iv) Locating these clones to the already detected methods, (v) Extraction of changes between every two consecutive revisions, and (vi) Reflecting these changes to the already detected methods and clones residing in these methods.

We extract methods using CTAGS[2]. For detecting method genealogies we follow the procedure proposed by Lozano and Wermelinger [LW08]. The genealogy of a particular method helps us to understand how a particular method evolved during software evolution. As we detect method genealogies, we also detect clone genealogies by locating the propagation of the clone fragments through the methods.

Clone Genealogy. By the term *clone genealogy* we mean the genealogy of a particular code fragment which is also regarded as a clone fragment by the clone detector. By detecting the genealogy of a particular clone fragment we can easily determine how that fragment was changed during the evolution. A particular clone class may contain two or more clone fragments. We determine a separate genealogy for each of these clone fragments.

We use NiCad clone detector for detecting and extracting clones from each revision of a subject system. The main purpose of choosing NiCad is that it can detect clones of different clone-types separately including Type 3 with high precision and recall [RC09, RCK09]. For detecting Type 3 clones, we considered a dissimilarity threshold of 20% with blind renaming of identifiers. For the details of the preliminary steps mentioned above and for NiCad setup, we refer the interested readers to our earlier work [MRS12a]. Here, we should

[2] Ctags: `http://sourceforge.net/projects/ctags/?source=directory`

note that before using the NiCad outputs for Type-2 and Type-3 cases, we pre-processed them in the following way.

(1) Every Type-2 clone class that exactly matched any Type-1 clone class was excluded from Type-2 outputs.

(2) Every Type-3 clone class that exactly matched any Type-1 or Type-2 class was excluded from Type-3 outputs.

We performed these because we wanted to investigate each of the three types of clones separately. The above two pre-processing steps ensure that the set of Type 2 clone classes that we investigate does not contain any Type 1 clone class. Also, the set of our investigated Type 3 clone classes does not contain any Type 1 or Type 2 clone classes. The detection mechanism of late propagation clone-pairs is described in the following subsection.

3.2 Detection of Clone-Pairs that Experienced Late Propagation

After completing the preliminary steps described above we automatically mine the late propagation clone-pairs. At the very beginning, we assume a global list of clone pairs each of which has the potential of experiencing late propagation. We call such a clone pair a CPLP (Clone Pair having the potential of experiencing Late Propagation). We call this list the Global List.

Clone Pair having the potential of experiencing Late Propagation (CPLP). We consider a pair of code fragments, (*CF1* and *CF2*), which are clones of each other in revision R_i. A commit operation C_i was applied on R_i and one or both of these fragments changed. However, because of this change, *CF1* and *CF2* were not considered as clones of each other in revision R_{i+1}. In other words, the change in C_i is a diverging change for the pair (*CF1*, *CF2*). This pair is considered as a CPLP because, there is a possibility that in a future commit operation, the fragments *CF1* and *CF2* will converge (i.e., *CF1* and *CF2* will again be considered as clones of each other).

The Global List remains empty initially. We examine the commit operations sequentially from the very beginning one. We only consider those commits where there were changes to one or more clone fragments of a particular clone type. As we examine the commit operations, we update the Global List and mark some clones pairs (i.e., some CPLPs) in this list as the late propagation clone pairs. Suppose, C_i is such a commit which was applied on revision R_i and the immediate next revision R_{i+1} was created as a result. We perform the following steps sequentially considering C_i.

Step 1. Determining the list of affected clone fragments. We identify the list of clone fragments (in revision R_i) that received some changes during C_i. We call this list the List of Affected Clone Fragments.

Step 2. Determining the list of affected clone pairs. We make a list of clone pairs that involve one or more clone fragments in the List of Affected Clone Fragments. We denote this list of clone pairs as the List of Affected Clone Pairs.

Step 3. Updating the Global List using the List of Affected Clone Fragments. We identify those clone pairs in the Global List each of which involves any of the clone fragments in the List of Affected Clone Fragments. There is a possibility that such a clone pair in the Global List has converged. In order to check this we determine whether the fragments in such a pair are considered as clones of each other in revision R_{i+1} which was created because of commit C_i. If this is true, then we understand that this clone pair in the Global List has experienced late propagation. We mark this clone pair as a late propagation pair.

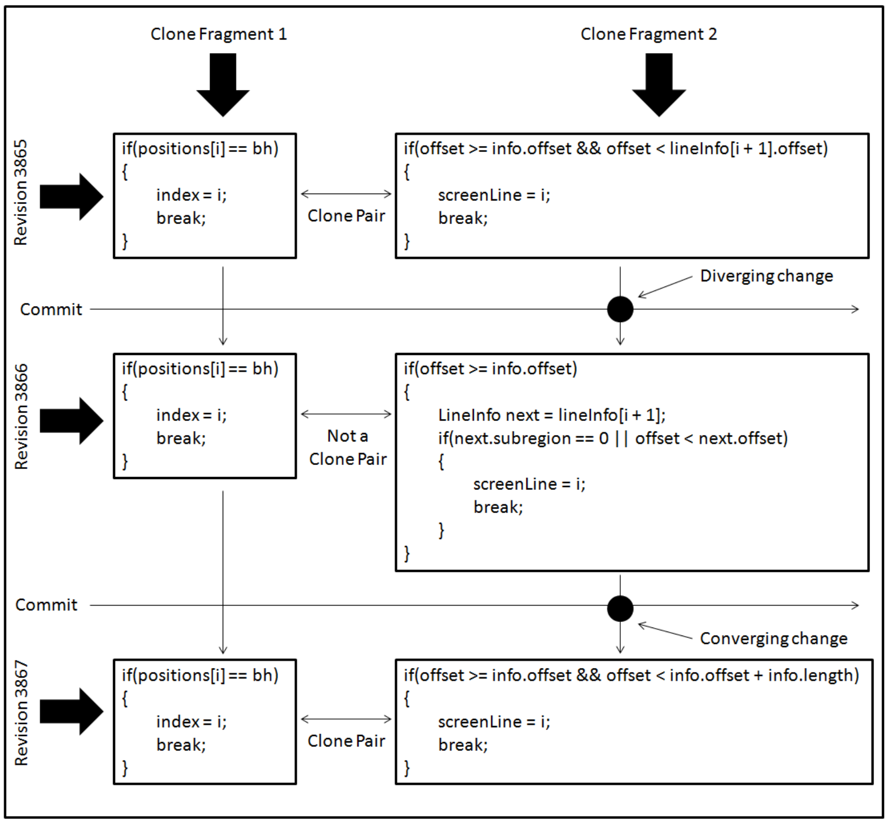
Step 4. Updating the Global List using the List of Affected Clone Pairs. If any pair in the List of Affected Clone Pairs already appears in the Global List, we do not need to consider this pair because, this has already been handled in the previous step. Considering the remaining pairs (in the List of Affected Clone Pairs), we determine the CPLPs (i.e., the clone pairs that have the potential of experiencing late propagation). If the two fragments in a remaining pair are not considered as clones of each other in revision R_{i+1}, then this pair is a CPLP. We include the CPLPs in the Global List.

For each of the commit operations we follow the above four steps, update the Global List and mark some CPLPs in this list as the late propagation pairs if they converge. After examining all the commit operations we get all the late propagation clone pairs of a particular clone type.

Now, let us assume that a particular pair in the Global List has been marked as a late propagation pair during the examination of the commit operation C_i. This pair has the following three possibilities during future evolution.

– The pair may again experience late propagation. In our experiment we detect each of the occurrences of late propagations a particular clone pair experienced during evolution.
– The fragments in the pair may evolve independently. However, independent evolution of such fragments without any convergence is not our concern in this research work.
– One or both fragments may form new pair(s) with other fragments of the same or other clone types. In this case, our implementation considers the new pairs in calculation because they can experience late propagation.

Detection of late propagation considering an individual clone-type. Suppose we are detecting late propagation considering the clones of Type j where j = 1, 2, or 3. The clone fragments *CF1* and *CF2* are clones of this type in revision R_i. Because of the commit C_i on revision R_i, the fragments *CF1* and *CF2* diverged. Let us assume that in commit C_{i+n}, the fragments converged and they were again considered as clones of Type j. Then, we consider this late propagation as a late propagation of Type j. It might be the case that after converging, *CF1* and *CF2* were not considered as clones of Type j. They were considered as clones of Type k where k= 1, 2, or 3 and $j \neq k$. In this case we do not consider a late propagation, because the fragments changed their types. While detecting late propagation in the clones of Type



```
if(positions[i] == bh)
{
    index = i;
    break;
}
```

```
if(offset >= info.offset && offset < lineInfo[i + 1].offset)
{
    screenLine = i;
    break;
}
```

```
if(positions[i] == bh)
{
    index = i;
    break;
}
```

```
if(offset >= info.offset)
{
    LineInfo next = lineInfo[i + 1];
    if(next.subregion == 0 || offset < next.offset)
    {
        screenLine = i;
        break;
    }
}
```

```
if(positions[i] == bh)
{
    index = i;
    break;
}
```

```
if(offset >= info.offset && offset < info.offset + info.length)
{
    screenLine = i;
    break;
}
```

Fig. 2 An example of late propagation in a Type 3 clone pair from subject system jEdit

k, the fragments *CF1* and *CF2* are considered to determine whether they experienced a late propagation of Type k. However, we plan to investigate the intensity of such mixed type late propagations (i.e., where the participating fragments were considered of one clone type before divergence but of another clone type after convergence) as a future work.

An example of late propagation in Type 3 clones. We present an example of late propagation that occurred to a Type 3 clone pair of our subject system jEdit. We automatically detect this late propagation by applying our late propagation detection tool. We present Fig. 2 for describing the late propagation example.

In Fig. 2 we see a Type 3 clone pair in revision 3865 of our candidate system jEdit. As we can see, the participating clone fragments (denoted as *Clone Fragment 1* and *Clone Fragment 2*) are two if-blocks. NiCad detects these Type 3 clones by considering a dissimilarity threshold of 20% and applying blind renaming of identifiers. These two clone fragments belong to two different source code files[3] [4]. The names of the container methods of these two clone fragments are *removePosition* and *getScreenLineForOffset* in revision 3865 . The commit operation applied on revision 3865 changed the clone fragment at the right hand side (i.e., Clone Fragment 2). Because of this change they were not considered as a clone pair in revision 3866. Thus, this change is a diverging change for the clone pair. However, the commit operation applied on revision 3866 again changed the fragment at the right hand side and the fragments converged (i.e., became a clone pair) in revision 3867. Thus, this clone pair experienced a late propagation.

4 Experimental Results and Discussion

We applied our implementation on each of the eight subject systems in Table 2 and identified the clone pairs that experienced late propagation considering each of the three clone types (Type 1, Type 2, Type 3). In the following subsections, we answer the research questions mentioned in the introduction by presenting and analyzing our experimental results. In our previous study [MRS14d] we did not consider late propagation between clone fragments remaining in the same method. We consider such late propagations in this extended study. Also, during the period of divergence of a particular late propagation, any one or both of the two participating code fragments (i.e., the code fragments that were considered as a clone-pair before divergence) might become non-clone fragments or be considered as clone fragments of different clone classes. We identify late propagations considering both of these cases in this extended research work.

4.1 **RQ 1: What percentage of late propagations in Type 3 clones mainly occur because of the changes in the non-matched portions of the participating clone fragments?**

Rationale. We know that Type 3 clones have both cloned and non-cloned portions. If a late propagation in Type 3 clones occur because of the changes in the non-matched (i.e., non-cloned) portions only, then this late propagation might not be important from the perspective of clone management. If it is observed that a significant portion of the late propagations in Type 3 clones occur because of the changes in the non-matched portions, then it is important to identify and discard these late propagations while doing investigations regarding clone management. We answer RQ 1 in the following way.

Methodology. Let us consider that a pair of Type 3 clone fragments has experienced a late propagation. If the matched portions of none of these two clone fragments were modified during the diverging and converging change, and also, during the period of divergence, then we decide that this late propagation

[3] Source code file for Clone Fragment 1: trunk/org/gjt/sp/jedit/buffer/OffsetManager.java
[4] Source code file for Clone Fragment 2: trunk/org/gjt/sp/jedit/textarea/ChunkCache.java

can be ignored when making clone management decisions. We at first detect all the occurrences of late propagations in Type 3 clones, and then determine which late propagations occurred only because of the changes in the non-matched portions. A particular clone pair can experience late propagation more than once. We detect and check all those for our investigation regarding RQ 1. We automatically check a late propagation in the following way.

Let us assume that two code fragments *CF1* and *CF2* are Type 3 clones of each other. They experienced a late propagation where the diverging change occurred in commit C_i and the converging change occurred in commit C_j ($C_j > C_i$). We check all these commits from C_i to C_j to determine whether any one or both of *CF1* and *CF2* changed in these commits and whether the changes occurred in the matched or non-matched portions of the fragments. Suppose, we are going to check the commit operation C where $C_i <= C <= C_j$. We extract the two instances of the two code fragments *CF1* and *CF2* before this commit. Let us assume that these instances are $CF1_{before}$ and $CF2_{before}$ respectively. We also determine the two instances, $CF1_{after}$ and $CF2_{after}$, after the commit. We blind-rename the instances $CF1_{before}$ and $CF2_{before}$ and then find the differences of these blind-renamed instances using *diff* command. From *diff* output we determine the matched and non-matched portions of $CF1_{before}$ and $CF2_{before}$. We then determine the differences between $CF1_{before}$ and $CF1_{after}$ using *diff* command to identify the changes occurred to $CF1_{before}$ in terms of additions, deletions, and modifications. We determine whether these changes occurred to matched or non-matched portions of $CF1_{before}$ using the line numbers of the changes. In the same way we determine whether any changes occurred to the matched or non-matched portions of $CF2_{before}$. If in each of the commit operations from C_i to C_j the two code fragments *CF1* and *CF2* received changes only in their non-matched portions, then we consider this late propagation as an ignorable one.

Considering the late propagations occurred to the Type 3 clones of each of the subject systems we determine what proportion of late propagations occurred only because of the changes in the non-matched portions and thus, are ignorable. Table 3 shows these proportions for our subject systems. We see that in case of five subject systems (Ctags, BRL-CAD, Freecol, Carol, and Java-ML) this percentage is zero. For the remaining three subject systems: Camellia, jEdit, and Jabref these percentages are 0.14%, 7.22%, and 0.07% respectively.

Answer to RQ 1: *According to our experimental results, the percentage of late propagations in Type 3 clones occurred only because of the changes in the non-matched portions of the participating clone fragments is very low (i.e., less than one) in most of the subject systems (i.e., seven out of eight systems) we have studied.* However, our analysis is based on only eight subject systems which are not enough to generalize our findings. The percentage of ignorable late propagations can be considerable for some systems (for example, 7.22% in case of our subject system jEdit). We should ignore these late propagations when making clone management decisions. Our implemented prototype tool can automatically detect such ignorable late propagations so that we can dis-

Table 3 Late Propagations in Type 3 Clones

	Ctags	**Camellia**	**BRL-CAD**	**Freecol**	**jEdit**	**Carol**	**Jabref**	**Java-ML**
LP	5	728	1593	1062	83	644	3028	65
LPNM	0	1	0	0	6	0	2	0
PLPNM	0%	0.14%	0%	0%	7.22%	0%	0.07%	0%

LP = Total number of late propagations in Type 3 clones
LPNM = Number of late propagations in Type 3 clones occurred only because of the changes in the non-matched portions
PLPNM = Percentage of late propagations in Type 3 clones occurred only because of the changes in the non-matched portions

Table 4 Statistics regarding late propagations in different clone-types

	Type 1				Type 2				Type 3			
System	CG	CPL	LG	LP	CG	CPL	LG	LP	CG	CPL	LG	LP
Ctags	146	0	0	0	170	0	0	0	549	5	6	5
Camellia	584	9	10	61	189	84	30	476	571	230	89	727
BRL-CAD	343	0	0	0	171	0	0	0	586	324	31	1593
Freecol	6593	232	47	3498	675	177	54	8903	4748	272	133	1062
jEdit	42778	14	21	14	1536	0	0	0	9756	50	68	77
Carol	1969	12	8	12	751	1	2	2	3022	225	133	644
Jabref	4262	7	9	7	895	3	4	3	5849	601	280	3026
Java-ML	429	4	5	4	404	33	19	110	1103	33	23	65

CG = Total number of clone genealogies
CPL = Total number of clone-pairs that experienced late propagation
LG = Number of distinct clone genealogies that experienced late propagation
LP = Total number of late propagations (discarding the ignorable ones in Type 3 case)

card them from considerations. For answering the remaining research questions we ignore these ignorable late propagations occurred in Type 3 clones.

4.2 **RQ 2: Are the intensities of late propagation different in different types of clones?**

Rationale. If it is observed that late propagation in a particular clone-type is more intense compared to the other clone-types, then it is an implication that clones of that particular clone type have a higher probability of introducing bugs and inconsistencies to the code-base compared to the other types. Thus, it would be beneficial if we could refactor clones of that particular type with higher priority. By minimizing these clones we can minimize the possibility of faults and inconsistencies to the code-base.

Methodology. For answering this research question we applied our prototype tool on each of the candidate systems and determine the following measures considering each of the three types of clones of each of the subject systems.

- The total number of clone genealogies
- The number of clone-pairs (i.e., pair of clone genealogies) that experienced late propagation
- The number of distinct clone genealogies that experienced late propagation
- The total number of late propagations occurred to the clone fragments. A particular pair of clone genealogies can experience late propagation more than once. We determine all of the occurrences of late propagations experienced by each clone pair.

We show these measures in Table 4. We investigate the intensity of late propagations in different clone types in the following two ways.

- **Investigation 1:** By determining and comparing the probability that a clone genealogy of a particular clone type will experience a late propagation.
- **Investigation 2:** By determining and comparing how often a pair of clone genealogies of a particular clone type experienced a late propagation.

Investigation 1: *Comparison of the probability that a clone genealogy of a particular clone type will experience a late propagation.*

We calculate probability as percentage. Considering each clone-type of each of the subject systems we determine the percentage of clone genealogies that experienced late propagation. These percentages are shown in Table 5. We calculate these percentages from Table 4.

From Table 5 we see that for five out of eight subject systems (except Camellia, Freecol, and Java-ML) Type 3 clones exhibit the highest intensity of late propagation in comparison with the other two clone types (Type 1, and Type 2). For three systems (i.e., Camellia, Freecol, and Java-ML), Type 2 clones exhibit the highest intensity.

Statistical significance test regarding the intensity of late propagation. We were also interested to investigate whether the intensity of late propagation in Type 3 clones is significantly higher than the intensity of late propagation in Type 1 or Type 2 clones. We perform our investigation using Mann-Whitney-Wilcoxon (MWW) tests. We have already determined the percentage of clone genealogies that experienced late propagation considering each clone type of each of the subject systems. These percentages are shown in Table 5. Thus, for a particular clone-type we get eight percentages from eight subject systems. We performed MWW tests [mwwb] for each pair of clone-types. For example, in case of the pair (Type 1, Type 3), we determine whether the eight percentages regarding Type 1 are significantly different than the eight percentages regarding Type 3. Here, we should note that MWW test is non-parametric and does not require the samples to be normally distributed [mwwa]. This test can be applied to both small and large sample sizes [mwwc]. In our research, we perform this test considering a significance level of 5%.

According to our MWW test result, the percentages regarding Type 3 case are significantly higher than the percentages regarding Type 1 case with *p-value* = 0.01 (approximately) for both one-tailed and two-tailed tests, and

Table 5 Percentage of clone genealogies that experienced late propagations from different clone-types

System	PCGLP - Type 1	PCGLP - Type 2	PCGLP - Type 3	CTHP
Ctags	0%	0%	1.09%	Type 3
Camellia	1.71%	15.87%	15.59%	Type 2
BRL-CAD	0%	0%	5.29%	Type 3
Freecol	0.71%	8%	2.8%	Type 2
jEdit	0.05%	0%	0.7%	Type 3
Carol	0.4%	0.27%	4.4%	Type 3
Jabref	0.21%	0.45%	4.79%	Type 3
Java-ML	1.16%	4.7%	2.09%	Type 2

PCGLP = Percentage of clone genealogies that experienced late propagations
CTHP = Clone-type with the highest percentage

with an effect size (r) of 0.71. We see that *p-value* < 0.05. Also, the effect size is large [effa]. The effect size calculation procedure for the MWW test is available on-line [effb]. Finally, we can say that the intensity of late propagation in Type 3 clones is significantly higher than the intensity of late propagation in Type 1 clones. However, we did not get significant differences for any of the other two pairs: (Type 1, Type 2) and (Type 2, Type 3).

Investigation 2: *Comparison of how often a clone-pair of a particular clone type will experience a late propagation.*

We perform this investigation considering the clone-pairs that experienced late propagations. Considering each clone-type of each of the subject systems we determine how many times a particular pair of clone fragments experienced late propagation on an average. Table 6 shows this average value for each clone-type of each of the subject systems. We see that for five out of eight subject systems (except Freecol, Camellia, and Java-ML), the average number of late propagations received by a Type 3 clone-pair is higher than the average number of late propagations experienced by a Type 1 or Type 2 clone-pair. For Camellia, Type 1 clone pairs exhibit the highest average. For both of the subject systems Freecol and Java-ML, the highest average values are exhibited by the Type 2 clone pairs.

Answer to RQ 2. *According to our investigation results, the intensity of late propagation in Type 3 clones is higher compared to the intensity of late propagation in the other two clone-types for five out of eight candidate systems. Also, according to the MWW test results, Type 3 clones exhibit a significantly higher intensity of late propagations than Type 1 clones. Thus, possibly Type 3 clones have a higher probability of introducing faults and inconsistencies to a code-base than the clones of the other two clone-types.*

Table 6 The average number of times a clone-pair experienced late propagation from different clone types

System	AT - Type 1	AT - Type 2	AT - Type 3	CTHAT
Ctags	0	0	1	Type 3
Camellia	6.77	5.66	3.16	Type 1
BRL-CAD	0	0	4.92	Type 3
Freecol	15.07	50.29	3.9	Type 2
jEdit	1	0	1.54	Type 3
Carol	1	2	2.86	Type 3
Jabref	1	1	5.03	Type 3
Java-ML	1	3.33	1.96	Type 2

AT = Average number of times a clone-pair experienced late propagations
CTHAT = Clone-type with the highest average number of times

4.3 RQ 3: Late propagations in which types of clones are more related to bugs?

Rationale. In a previous study Barbour et al. [BKZ13] found that late propagations in clones are related to bugs. However, they studied only Type 1 and Type 2 clones. Moreover, they did not report bugs for these two cases separately and thus, did not draw a comparative scenario between the bug-proneness of late propagations in Type 1 and Type 2 clones. We believe that understanding the comparative bug-proneness of the late propagations in three types of clones is important. If it is observed that the late propagations in a particular type of clones have a very low probability of being related to bugs compared to the other clone-types, then the late propagations of that particular clone-type could be ignored. In our study we investigate the bug-proneness of the late propagations in three types of clones (Type 1, Type 2, and Type 3) separately and show a comparative scenario considering these clone types.

Methodology. Previously Barbour et al. [BKZ13] performed a similar kind of investigation. They at first determined those clone pairs that experienced late propagations. Then, they determined whether any of these pairs ever experienced a fault fix during evolution. We perform a more in-depth investigation where we determine whether a fault fix occurred during the period of divergence of a particular late propagation. We believe that a particular late propagation occurred to a particular clone pair can only be related to a bug-fix if the bug-fix occurred during the late propagation period (i.e., the period of divergence). A pair of clones that experienced a late propagation can also experience a bug-fix however, the bug-fix might not occur during the period of late propagation. In this case we cannot relate this late propagation to the bug-fix.

We at first extract the SVN commit logs for each of the subject systems. The log contains the purpose why each of the revisions was created. If a revision was created because of a bug-fix, the corresponding log mentions it and generally includes the bug-ID. We identify the bug-fix commits from the commit log of a subject system using the heuristic proposed by Mockus and Votta [MV00]. Barbour et al. [BKZ13] also used the same heuristic in order to identify the bug-fix commits. As an example, if a commit message contains the word 'bug', then we consider the commit as a bug-fix commit. However, such an heuristic might cause false positives (i.e., identifying a commit as a bug-fix commit which was not actually done because of a bug-fix). According to the investigation results of Barbour et al. [BKZ13], this heuristic can help us detect bug-fix commits with a precision of 87%. We also perform manual investigations on the bug-fix commits detected from our candidate systems using this heuristic. From our analysis on 400 bug-fix commits (the first 50 bug-fix commits from each of the eight subject systems) we find that around 84% of these commits are true positives (i.e., are really bug-fix commits).

We detect late propagations considering each type of clones. A particular pair of clone fragments might experience late propagation more than once. In this investigation we detect all the late propagations that a particular clone pair experienced during evolution. A late propagation consists of a clone pair, a diverging commit, and a converging commit. We identify a late propagation to be related to a bug-fix if the following two conditions are satisfied: (1) at least one of the bug-fix commits (detected using our heuristic) occurred in between the diverging and converging commits of the late propagation, and (2) at least one clone fragment of the clone-pair was changed in this bug-fix commit. In this way we determine how many late propagations were related to bug fix. Considering each type of clones we determine the number of late propagations that were related to bug-fix. In case of Type 3 clones we disregard all those late propagations that occurred because of the changes in the non-matched portions of the clones. We compare the intensity of buggy late propagations in three types of clones in the following two ways.

– **Investigation 1:** By determining and comparing the possibility that a late propagation occurred to a clone-pair of a particular clone-type will be a buggy late propagation.
– **Investigation 2:** By determining and comparing the possibility that a clone fragment of a particular clone-type will experience a buggy late propagation.

Investigation 1: Comparison of the possibility that a late propagation occurred to a clone-pair of a particular clone-type will be a buggy late propagation.

Considering each clone-type of the each of the subject systems we determine the total number of late propagations, and the number of late propagations that are related to bug-fix. The percentage of buggy late propagations (i.e., the late propagations related to bug-fix) for each clone-type of each candidate system is shown in Table 7. Here we should note that more than one late propagations might be related to the same bug-fix. In case of Type 3 clones we consider only those late propagations that occurred because of the changes in matched portions of the clone fragments.

Table 7 Percentage of late propagations related to bug-fix

	Type 1		Type 2		Type 3		
System	**LP**	**PLPBF**	**LP**	**PLPBF**	**LP**	**PLPBF**	**TCHP**
Ctags	0	0%	0	0%	5	0	n/a
Camellia	61	3.28%	476	22.9%	727	15.13%	Type 2
BRL-CAD	0	0%	0	0%	1593	22.15%	Type 3
Freecol	3498	35.79%	8903	40.08%	1062	39.64%	Type 2
jEdit	14	71.43%	0	0%	77	70.13%	Type 1
Carol	12	0%	2	0%	644	34.63%	Type 3
Jabref	7	0%	3	0%	3026	41.21%	Type 3
Java-ML	4	0%	110	29.09%	65	36.92%	Type 3

LP = Total number of late propagation(s).
PLPBF = Percentage of late propagations that experienced bug-fix.
TCHP = The type of clones that experienced the highest proportion of bug-fix late propagations.

From Table 7 we see that Type 1 and Type 2 clones of Ctags and BRL-CAD, and also, Type 2 clones of jEdit did not experience any late propagations. In this table we also show the type of clones that experienced the highest proportion of bug-fix late propagations. We see that for four systems (BRL-CAD, Carol, Jabref, and Java-ML) out of eight subject systems, Type 3 clones experienced the highest proportion of buggy late propagations. In case of two subject systems (Camellia, Freecol) of the remaining ones, Type 2 clones experienced the highest proportion of bug-fix late propagations. In case of jEdit, Type 1 clones received the highest percentage of buggy late propagations. Thus, we see that for most of the subject systems Type 3 clones experienced the highest percentage of buggy late propagations.

Investigation 2: Comparison of the possibility that a clone fragment of a particular clone-type will experience a buggy late propagation.

Considering each type of clone of each of our candidates systems we determine the total number of clone genealogies created during system evolution, and the number of clone genealogies that experienced a late propagation related to a bug-fix. The percentage of these buggy late propagation genealogies with respect to all clone genealogies in case of each clone-type of each of the candidates systems is shown Table 8. In case of each of the subject systems, the table also shows the clone-type from which the highest proportion of clone genealogies experienced buggy late propagations.

From Table 8 we see that none of the three clone-types in Ctags received late propagations that are related to bug-fix. For most of the remaining subject systems (four systems out of seven), the proportion of Type 3 clones that experienced buggy late propagations is the highest compared to the proportions regarding the other two clone-types.

Statistical significance tests regarding the probability of experiencing buggy late propagation. We also wanted to investigate whether

Table 8 Percentage of clone genealogies that experienced buggy late propagations

	Type 1		Type 2		Type 3		
System	**CG**	**PGBL**	**CG**	**PGBL**	**CG**	**PGBL**	**CTHP**
Ctags	146	0%	170	0%	549	0%	n/a
Camellia	584	0.51%	189	10.05%	571	8.58%	Type 2
BRL-CAD	343	0%	171	0%	586	4.77%	Type 3
Freecol	6593	0.71%	675	5.93%	4748	2.10%	Type 2
jEdit	42778	0.04%	1536	0%	9756	0.56%	Type 3
Carol	1969	0%	751	0%	3022	2.66%	Type 3
Jabref	4262	0%	895	0%	5849	4.07%	Type 3
Java-ML	429	0%	404	3.21%	1103	0.73%	Type 2

CG = Total number of clone genealogies of a particular clone-type.
PGBL = Percentage of genealogies that experienced a buggy late propagation.
CTHP = The clone-type where the highest proportion of clone genealogies experienced buggy late propagations.

Type 3 clones have a significantly higher probability of experiencing buggy late propagations compared to Type 1 and Type 2 clones. Considering each clone type for each of the candidate systems we determined the percentage of clone-genealogies that experienced buggy late propagations. These percentages are recorded in Table 8. We perform MWW tests [mwwb] to determine whether the percentages regarding Type 3 case are significantly higher than the percentages regarding Type 1 and Type 2 cases. We perform the tests considering the significance level of 5%. According to our test results, the percentages for Type 3 case are significantly higher than the percentages for Type 1 case with a *p-value* < 0.01 for both two-tailed test and one-tailed test, and with an effect size of 0.67. We see that the *p-value* is smaller than 0.05. Thus, we can say that Type 3 clones exhibit a significantly higher probability of experiencing buggy late propagations compared to Type 1 clones. However, there is no significant difference between probabilities regarding Type 2 and Type 3 clones and also, between the probabilities for Type 1 and Type 2 clones.

Answer to RQ 3. *From our investigations we can state that Type 3 clones have a higher possibility of experiencing buggy late propagations compared to the clone fragments of the other two clone-types. Moreover, the probability of experiencing buggy late propagations for Type 3 clones is significantly higher compared to Type 1 clones.*

4.4 RQ 4: Clones of which clone-type(s) have higher possibilities of experiencing bug-fixing changes at the time of convergence?

In the previous research question we investigated those late propagations each of which experienced a bug-fix. A bug-fix can occur at any commit operation during the period of late propagation. However, we believe that it is important to know whether the bug-fix occurred at the time of convergence (i.e., at the converging commit) or not. If a bug-fix commit affects two previously diverged clone fragments in such a way that they converge together, then we can understand that for fixing of the bug it was necessary to ensure consistency of the diverged clone fragments. Thus, these clone fragments might be considered for management with high priority. If not refactorable, then these clone fragments should always be tracked to maintain their consistency. The existing clone tracker *CloneTracker* [DR07, DR08] does not automatically track all the clone fragments in a subject system. It allows programmers to select a subset of clones for tracking. Moreover, *CloneTracker* does not prioritize clones for tracking. Thus, a programmer is responsible to infer the more important clones for tracking and let *CloneTracker* know about this. In such a situation automatic prioritization of clone fragments for tracking can help programmers a lot. Our implemented system can automatically identify those clone fragments that received bug-fixing changes at the converging commit operation. Possibly these clone fragments can be prioritized for tracking. Here, we should note that in this research we only investigate the past evolution history of the clone fragments. This might not be the case that a clone fragment that experienced bug-fixing changes during a late propagation at past will also experience bugs in the future. We would like to investigate the likeliness of occurrence of such a phenomenon as a future work. In RQ 4 we investigate whether clone fragments experience bug-fixing changes at the time of convergence, and if so, how the intensity of this phenomenon differs across clone-types. We answer RQ 4 in the following way.

Methodology. We at first select all those late propagations each of which experienced a bug-fix using the methodology described in the previous research question. Then we automatically check each of these late propagations to determine whether the bug-fix occurred at the converging commit (i.e., the commit operation where two previously diverged clone fragments converged because of the changes in any one or both of the fragments). We determine the number of clone fragments that experienced such late propagations. Table 9 shows the percentage of clone fragments that experienced this type of late propagations with respect to all clone fragments in each clone-type of each of the candidate systems.

From Table 9 we see that in case of most of the subject systems disregarding Ctags, the proportion of clones that experienced bug-fixing changes at the time of convergence is the highest in Type 3 case. We disregard Ctags because none of the clone fragments in Ctags experienced bug-fixing changes.

Statistical significance test regarding the possibility of experiencing bug fixing changes at the time of convergence. As we have done previously, we wanted to investigate whether Type 3 clones exhibit a significantly higher possibility of experiencing bug-fixing changes at the time of convergence compared to Type 1 and Type 2 clones. We perform MWW tests [mwwb] to determine whether the percentages of Type 3 clone-genealogies experiencing bug-fixing changes at the time of convergence are significantly different than the corresponding percentages for Type 1 and Type 2 case. According to our tests considering a significance level of 5%, the percentages regarding Type 3 case are significantly higher than the percentages regarding Type 1 case with

Table 9 Percentage of clone genealogies that experienced bug-fixing changes at the time of convergence

	Type 1		Type 2		Type 3		
System	**CG**	**PCGB**	**CG**	**PCGB**	**CG**	**PCGB**	**CTHP**
Ctags	146	0%	170	0%	549	0%	n/a
Camellia	584	0%	189	9.52%	571	7.9%	Type 2
BRL-CAD	343	0%	171	0%	586	3.92%	Type 3
Freecol	6593	0.71%	675	4.59%	4748	1.58%	Type 2
jEdit	42778	0.03%	1536	0%	9756	0.42%	Type 3
Carol	1969	0%	751	0%	3022	1.16%	Type 3
Jabref	4262	0%	895	0%	5849	3.54%	Type 3
Java-ML	429	0%	404	3.21%	1103	0.73%	Type 2

CG = Total number of clone genealogies of a particular clone-type.
PCGB = Percentage of clone genealogies that received bug-fixing changes at the time of convergence.
CTHP = The clone-type where the highest proportion of clone genealogies experienced bug-fixing changes at the time of convergence.

p-value < 0.01 for both one-tailed and two-tailed tests, and with an effect size of 0.68. We see that the *p-value* is less than 0.05, and also, the effect size is large [effa]. Thus, we can say that Type 3 clones exhibit a significantly higher possibility of experiencing bug-fixing changes at the time of convergence compared to Type 1 clones. However, the difference between the percentages regarding the Type 2 and Type 3 cases is not statistically significant. The same is true for the Type 1 and Type 2 cases.

Answer to RQ 4: According to our investigation, *Type 3 clones have higher possibilities of experiencing bug-fixing changes at the time of converging compared to the clone fragments in each of the other two clone types. Moreover, Type 3 clones exhibit a significantly higher probability of experiencing bug-fixing changes at the converging commits compared to Type 1 clones.* We have already discussed that the clone fragments that experience bug-fixing changes at the time of convergence should be considered important for tracking. Our implemented prototype tool can automatically identify such clone fragments by analyzing clone evolution history and thus, can help programmers identify the important tracking candidates while dealing with *CloneTracker*.

4.5 **RQ 5: Do the participating clone fragments in a clone pair that experience late propagation generally remain in different files?**

Rationale. According to a number of studies [DLL09, VPV10], the program entities that often need to be changed together (i.e., that often require corresponding changes) should remain in close proximity to each other so that while changing a particular entity the developer does not miss to look at other entities that may require corresponding changes. Considering this fact we suspect that possibly the clone fragments in a clone pair that exhibit late propagation

Table 10 Percentage of late propagation clone pairs each having clone fragments from the same file or from different files

	Type 1				Type 2				Type 3			
System	**CPL**	**SF**	**DF**	**MLP**	**CPL**	**SF**	**DF**	**MLP**	**CPL**	**SF**	**DF**	**MLP**
Ctags	0	0%	0%	n/a	0	0%	0%	n/a	5	100%	0%	S
Camellia	9	44.4%	55.6%	D	84	19%	81%	D	230	39.6%	60.4%	D
BRL-CAD	0	0%	0%	n/a	0	0%	0%	n/a	324	100%	0%	S
Freecol	232	94.8%	5.2%	S	177	93.2%	6.8%	S	272	46.3%	53.7%	D
jEdit	14	14.3%	85.7%	D	0	0%	0%	n/a	50	14%	86%	D
Carol	12	100%	0%	S	1	100%	0%	S	225	20.9%	79.1%	D
Jabref	7	100%	0%	S	3	0%	100%	D	601	26.6%	73.4%	D
Java-ML	4	0%	100%	D	33	87.9%	12.1%	S	33	81.8%	18.2%	S

CPL = Total number of clone pairs that experienced late propagations.
SF = Percentage of late propagation pairs each having clone fragments from the same file
DF = Percentage of late propagation pairs each having clone fragments from different files
MLP = The situation that occurs for most of the late propagation pairs. This filed can have either of the two values: 'S' or 'D'
S = For most of the late propagations, the two clone fragments belong to the same file
D = For most of the late propagations, the two clone fragments belong to different files

generally remain in two different files and as a result, the developers often forget to make corresponding changes to these clone fragments. We investigate in the following way to look into this matter.

Methodology. We have already said that considering each of the clone types of each of the subject systems we identify the clone pairs that experienced late propagation. For each of these pairs we determined whether the participating clone fragments remain in different files or in the same file. We determined two percentages - (i) the percentage of the clone pairs having clone fragments from different source code files and (ii) the percentage of clone pairs consisting of clone fragments from the same file. These percentages are shown in Table 10.

Analysis. From Table 10 we see that for ten cases (for example Type 1 case of jEdit, Type 2 case of Jabref) the percentage of late propagation clone pairs each having clone fragments from different files is higher than the percentage of late propagation clone pairs each having clone fragments from the same file. However, the opposite is true for nine cases (for example Type 1 case of Freecol, Type 1 case of Carol). The clone fragments in the remaining five cases (Type 1 and Type 2 cases of Ctags and BRL-CAD, and Type 2 case of jEdit) did not experience any late propagations.

Answer to RQ 5. From our investigation we understand that whether the two participating clone fragments in a particular clone pair remain in different files or in the same file, the clone pair can experience late propagation. Proximity of the constituent clone fragments in a clone pair possibly does not have any significant effect on the occurrence of late propagations to that pair.

Table 11 Percentage of late propagation clone pairs each consisting of method clones or block clones

	Type 1				Type 2				Type 3			
System	**CPL**	**BC**	**MC**	**MLP**	**CPL**	**BC**	**MC**	**MLP**	**CPL**	**BC**	**MC**	**MLP**
Ctags	0	0%	0%	n/a	0	0%	0%	n/a	5	100%	0%	B
Camellia	9	100%	0%	B	84	100%	0%	B	230	99.1%	0.87%	B
BRL-CAD	0	0%	0%	n/a	0	0%	0%	n/a	324	100%	0%	B
Freecol	232	100%	0%	B	177	100%	0%	B	272	97.1%	2.9%	B
jEdit	14	57.1%	42.9%	B	0	0%	0%	n/a	50	90%	10%	B
Carol	12	100%	0%	B	1	100%	0%	B	225	95.6%	4.4%	B
Jabref	7	100%	0%	B	3	0%	100%	M	601	98.2%	1.8%	B
Java-ML	4	0%	100%	M	33	87.9%	12.1%	B	33	90.9%	0.1%	B

CPL = Total number of clone pairs that experienced late propagations.
BC = Percentage of late propagation pairs each consisting of at least one block clone
MC = Percentage of late propagation pairs each consisting of method clones only
MLP = The situation that occurs for most of the late propagation pairs. This filed can have either of the two values: 'B' or 'M'
B = For most of the late propagation pairs, at least one of the two clone fragments is a block clone
M = For most of the late propagation pairs, both clone fragments are method clones

4.6 **RQ 6: Do block clones or method clones exhibit higher intensity of late propagation?**

Rationale. Intuitively, copying a block of statements from one method and pasting that block to several other methods is more difficult compared to copy-pasting a whole method. While pasting a block of statements into a method, the variable names and data types in the block might need to be changed in accordance with the variables and data types in that method. If there is a problem in making such correspondence and as a result, the variables are not changed correctly, then this will create inconsistency in future evolution. If a number of block clones (forming a clone class) are created with such inconsistencies, these inconsistencies in different clone fragments will be discovered at different times during evolution and as a result, late propagation will happen. Also, blocks might not have well defined boundaries as of methods. For this reason, keeping track of block clones might seem to be more difficult compared to method clones to a programmer.

Methodology. We perform the following two investigations for answering this research question.

– **Investigation 1:** *Investigating what proportions of the late propagations involve block-clones or method-clones.*
– **Investigation 2:** *Investigating what proportions of the block-clones and method-clones experienced late propagation.*

Investigation 1: *Investigating what proportions of the late propagations involve block-clones or method-clones.*

Considering each of the clone types of each of the subject systems, we at first determine those clone pairs that exhibited late propagation and then we determine whether the clone fragments in such a pair are block clones or method clones. We calculate: (i) the percentage of late propagation clone pairs each consisting of at least one block clone, and (ii) the percentage of late propagation clone pairs consisting of method clones only. These percentages regarding each clone-type of each subject system are shown in Table 11.

Analysis. From Table 11 we see that for almost all of the cases, the percentage of late propagation clone pairs involving block clones is much higher (100 % in many cases such as Type 2 case of Freecol) than the percentage of late propagation pairs consisting of only method clones. Although we determine the percentage of late propagation clone pairs having at least one block clone, for most of the cases we observed that both of the clones in such a pair are block clones. However, for a very few cases (such as Type 1 and Type 2 cases of Ctags) we did not get any clone pair experiencing late propagation.

Investigation 2: *Investigating what proportions of the block-clones and method-clones experienced late propagation.*

Considering each clone-type of each of the subject systems we first identify all the clone genealogies created during evolution, and then separate those into two disjoint subsets: (1) block clone genealogies, and (2) method clone genealogies. We also determine which of these block clone genealogies as well as which of the method clone genealogies experienced late propagations. Finally, we calculate the following two percentages:

– The percentage of block clones (i.e., block clone genealogies) that experienced late propagation with respect to all block clones.
– The percentage of method clones that experienced late propagation with respect to all method clones.

Table 12 shows the following four measures considering each clone-type of each of the subject systems: (1) the number of block clones, (2) the number of block clones that experienced late propagation, (3) the number of method clones, and (4) the number of method clones that experienced late propagations during evolution. Table 13 shows the percentages of block clones as well as method clones that experienced late propagations with respect to all block clones and method clones respectively. The percentages in this table were calculated from the values in Table 12. Table 13 shows that for most of the cases, the percentage of method clones that experienced late propagations is smaller compared to the corresponding percentage of block clones. We found only two cases (i.e., Type 1 case of Java-ML, and Type 2 case of Jabref) where the percentage of method clones that experienced late propagations is higher than the corresponding percentage of block clones.

Statistical Significance Tests. We wanted to determine whether the percentages of block clones that experienced late propagations are significantly higher than the corresponding percentages of method clones. Table 13 contains 24 cases (8 systems $\times$ 3 clone-types) in total. We perform Mann-Whitney-Wilcoxon tests [mwwb] to determine whether the percentages regarding block

Table 12 The number of block clones and method clones that experienced late propagations

	Type 1				Type 2				Type 3			
System	**NB**	**NBL**	**NM**	**NML**	**NB**	**NBL**	**NM**	**NML**	**NB**	**NBL**	**NM**	**NML**
Ctags	85	0	61	0	99	0	71	0	336	6	213	0
Camellia	398	10	186	0	177	30	12	0	520	86	51	3
BRL-CAD	311	0	32	0	153	0	18	0	520	31	66	0
Freecol	5721	47	872	0	508	54	167	0	3545	116	1203	18
jEdit	16828	10	25950	11	767	0	769	0	6437	46	3319	22
Carol	907	8	1062	0	296	2	455	0	1728	112	1294	21
Jabref	2262	9	2000	0	640	0	255	4	3797	246	2052	36
Java-ML	224	0	205	5	240	13	164	6	553	18	550	5

NB = Number of block clone genealogies created during evolution.
NBL = Number of block clone genealogies that experienced late propagation.
NM = Number of method clone genealogies created during evolution.
NML = Number of method clone genealogies that experienced late propagation.

Table 13 Percentage of block clones and method clones that experienced late propagations

	Type 1			Type 2			Type 3		
System	**PBC**	**PMC**	**HP**	**PBC**	**PMC**	**HP**	**PBC**	**PMC**	**HP**
Ctags	0%	0%	n/a	0%	0%	n/a	1.79%	0%	PBC
Camellia	2.51%	0%	PBC	16.94%	0%	PBC	16.53%	5.88%	PBC
BRL-CAD	0%	0%	n/a	0%	0%	n/a	5.96%	0%	PBC
Freecol	0.82%	0%	PBC	10.62%	0%	PBC	3.27%	1.49%	PBC
jEdit	0.06%	0.04%	PBC	0%	0%	n/a	0.71%	0.66%	PBC
Carol	0.88%	0%	PBC	0.67%	0%	PBC	6.48%	1.62%	PBC
Jabref	0.39%	0%	PBC	0%	1.56%	PMC	6.47%	1.75%	PBC
Java-ML	0%	2.43%	PMC	5.42%	3.65%	PBC	3.25%	0.91%	PBC

PBC = Percentage of block clones that experienced late propagation.
PMC = Percentage of method clones that experienced late propagation.
HP = The larger one of the above two percentages. This field can have either of the two values: PBC, and PMC.

clones in these cases are significantly higher compared to the percentages regarding method clones. We consider a significance level of 5%. According to our test, the percentages of block clones that experienced late propagations are significantly higher than the corresponding percentages of method clones with *p-value* = 0.02 for two-tailed test and 0.01 for one-tailed test, and with an effect size of 0.32. We see that the *p-value*s are smaller than 0.05, and thus, the percentages of block clones that experienced late propagations are significantly higher than the corresponding percentages of method clones.

Answer to RQ 6: *The clone pairs that experience late propagation generally consist of block clones instead of method clones.* Our second investigation shows that *block clones exhibit a significantly higher tendency of experiencing late propagation than method clones.* Such an observation implies that block clones possibly have higher probability of introducing inconsistencies to a codebase compared to the method clones. Thus, creating block clones is more risky

Table 14 Statistics regarding SPCP clones and late propagations

	Type 1			Type 2			Type 3		
System	**SPCP**	**LP**	**LPSPCP**	**SPCP**	**LP**	**LPSPCP**	**SPCP**	**LP**	**LPSPCP**
Ctags	26	0	n/a	16	0	n/a	118	5	3
Camellia	64	61	61	21	476	216	78	727	642
BRL-CAD	80	0	n/a	17	0	n/a	111	1593	1559
Freecol	143	3498	3498	94	8903	8885	180	1026	598
jEdit	486	14	14	21	0	n/a	128	77	25
Carol	84	12	12	110	2	2	498	644	336
Jabref	50	7	7	84	3	3	544	3026	2111
Java-ML	119	4	4	34	110	104	355	65	59

SPCP = The number of SPCP clone fragments.
LP = The total number of late propagations
LPISPCP = The number of late propagations involving SPCP clones

than creating method clones. We should possibly consider refactoring (if possible) block clones with higher priority.

4.7 RQ 7: Do late propagations mainly occur to the SPCP clones or non-SPCP clones?

Rationale. In a previous study [MRS14b] we showed that SPCP (Similarity Preserving Change Pattern) clones are the most important ones from the perspectives of clone management (such as clone tracking or refactoring). We suggested to mainly focus on managing SPCP clones when taking clone management decisions [MRS14b, MRS14a]. In this research question (i.e., RQ 7) we investigate whether late propagations mostly occur to the SPCP clones. In such a case we can say that proper management of SPCP clones (i.e., through refactoring or tracking) can help us minimize late propagations. We perform our investigation in the following two ways.

– **Investigation 1:** By investigating what proportions of late propagations occur to the SPCP clones.
– **Investigation 2:** By investigating the frequency of the occurrences of late propagations to the SPCP clones and non-SPCP clones.

Investigation 1: *Investigation on the proportion of late propagations occurred to the SPCP clones.*

If it is observed that most of the late propagation occur to the SPCP clones rather than non-SPCP clones, then we can decide that managing SPCP clones through refactoring and/or tracking can help us minimize the occurrences of late propagations considerably. We investigate in the following way.

We at first determine the SPCP clones in the code-base by applying our detection mechanism elaborated in our previous study [MRS14a]. Then, we determine all the occurrences of late propagations. We automatically check

Table 15 Percentage of late propagations involving SPCP clones

System	PLS - Type 1	PLS - Type 2	PLS - Type 3
Ctags	0%	0%	60%
Camellia	100%	45.38%	88.31%
BRL-CAD	0%	0%	97.86%
Freecol	100%	99.8%	58.28%
jEdit	100%	0%	32.48%
Carol	100%	100%	52.17%
Jabref	100%	100%	69.76%
Java-ML	100%	94.54%	90.76%

PLS = Percentage of late propagations involving SPCP clones
OPPS = Overall percentage per system

each of these late propagations and determine which late propagations involve SPCP clones (i.e., any of the two participating clone fragments in the late propagation are SPCP clones). Considering each clone-type of each of the candidate systems we determine how many of the corresponding late propagations involve SPCP clones. Table 14 shows the total number of SPCP clones, total number of late propagations, and the number of late propagations that involve SPCP clones. The percentage of late propagations involving SPCP clones considering each clone-type of each of the candidate systems is shown in Table 15.

From Table 15 we see that the percentage of late propagations involving SPCP clones is zero for Type 1 and Type 2 cases of Ctags and BRL-CAD, and also, for Type 2 case of jEdit. The reason is that we did not get any late propagations for these cases. This is also evident from Table 14. However, from this table (i.e., Table 14) we see that in each of these cases we found SPCP clones. Table 15 shows that in case of each of the subject systems, all of the late propagations in Type 1 clones involved SPCP clones. The same is true for Type 2 cases of most of the subject systems except Camellia. From Table 15 and 14 we understand that a number of late propagations in Type 2 and Type 3 cases do not involve SPCP clones. In Section 2 we explained that two clone fragments might experience late propagation(s), however, they will not be regarded as SPCP clones if they finally diverge and do not converge again. The late propagations that do not involve SPCP clones were experienced by such non-SPCP clone pairs.

Considering all clone types of all the candidate systems we found 20253 occurrences of late propagations in total, and 18139 of these involved SPCP clones. Thus, around 89.56% of the total late propagations involved SPCP clones. Such a finding implies that late propagations mainly occur to the SPCP clones. From this we come to the decision that we can considerably minimize the future occurrences of late propagations by managing the SPCP clones through refactoring and tracking.

Investigation 2: *Investigation on the frequency of the occurrences of late propagations to the SPCP clones and non-SPCP clones.*

Table 16 Frequency of late propagations in SPCP and Non-SPCP clones

	Type 1			Type 2			Type 3		
System	**FLS**	**FLNS**	**HF**	**FLS**	**FLNS**	**HF**	**FLS**	**FLNS**	**HF**
Ctags	0	0	n/a	0	0	n/a	1	1	n/a
Camellia	6.78	0	FLS	2.27	6.87	FLNS	2.89	3.73	FLNS
BRL-CAD	0	0	n/a	0	0	n/a	4.87	8.5	FLNS
Freecol	15.07	0	FLS	54.18	1.38	FLS	5.43	2.86	FLS
jEdit	1	0	FLS	0	0	n/a	1.77	1.35	FLS
Carol	1	0	FLS	2	0	FLS	2.52	3.34	FLNS
Jabref	1	0	FLS	1	0	FLS	6.04	3.63	FLS
Java-ML	1	0	FLS	3.59	1.5	FLS	2.19	1	FLS

FLS = Frequency of late propagations in SPCP clones.
FLNS = Frequency of late propagations in non-SPCP clones.
HF = Higher Frequency. This field can have two values: FLS or FLNS

From the previous investigation we understand that late propagations mainly occur to the SPCP clones rather than the non-SPCP clones. In this investigation we determine whether late propagations are more frequent to the SPCP clones or to the non-SPCP clones. If the frequency of late propagations to the SPCP clones is higher compared to the frequency of late propagations to the non-SPCP clones, then we can again decide that refactoring and tracking of SPCP clones can help us minimize late propagations. We perform our investigation in the following way.

Considering each clone-type of each of the candidate systems we determine the following four measures:

- **Measure 1:** The number of distinct clone pairs that involve SPCP clones and experienced late propagations,
- **Measure 2:** The number of distinct clone pairs that do not involve SPCP clones and experienced late propagations,
- **Measure 3:** The total number of late propagations each involving SPCP clone(s), and
- **Measure 4:** The total number of late propagations involving only non-SPCP clones.

We then determine the following two frequencies from the above measures.

- The frequency of late propagations in SPCP clones by dividing **Measure 3** by **Measure 1** .
- The frequency of late propagations in the non-SPCP clones by dividing the fourth measure (**Measure 4**) by the second one (**Measure 2**).

These frequencies for each clone-type of each of the subject systems are shown in Table 16. The table shows that for most of the cases (i.e., except Type 2 and Type 3 cases of Camellia, Type 3 case of Carol, and Type 3 case of BRL-CAD) the frequency of late propagations in SPCP clones is higher than the frequency of late propagations in non-SPCP clones.

Statistical significance test regarding the frequency of late propagations in SPCP and non-SPCP clones: We also wanted to investigate whether the frequency of late propagations in SPCP clones is significantly higher than the frequency of late propagations in non-SPCP clones. If we consider all three clone types of all eight candidate systems we get 24 cases (3 clone-types x 8 candidate systems) in total. We have already mentioned that we did not get any late propagations for Type 1 and Type 2 cases of Ctags and BRL-CAD, and Type 2 case of jEdit. Considering the remaining 19 cases we determine two sets of frequencies. One set contains the frequencies of late propagations in SPCP clones in these 19 cases. The other set contains the frequencies of late propagations in non-SPCP clones. We perform the Mann-Whitney-Wilcoxon test [mwwb] to determine whether the samples in these two sets are significantly different. We consider a significance level of 5%. According to the test results, the difference between these two sets is significant with *p-value* (probability value) $= 0.04$ (< 0.05) for two-tailed test and 0.02 for one-tailed test, and with an effect size of 0.33. We see that the *p-value*s are smaller than 0.05. Thus, we can say that the frequency of late propagations in SPCP clones is significantly higher than the frequency of late propagations in non-SPCP clones.

Answer to RQ 7: *According to our investigations we can state that most of the late propagations (around 89.56%) involve SPCP clones. In other words, late propagations mainly occur to the SPCP clones. Also, the frequency of the occurrence of late propagations in SPCP clones is significantly higher compared to non-SPCP clones. These findings imply that by managing SPCP clones through refactoring and tracking we can minimize the occurrences of late propagations considerably.* Moreover, SPCP clones not only include those clone fragments that experienced late propagations but also other clone fragments that are important to be updated consistently [MRS14b]. Thus, we should primarily focus on managing the SPCP clones. Management of SPCP clones through refactoring and tracking will not only help us minimize late propagations but also will help us in better maintenance of software systems.

4.8 Discussion

In our research we answered seven research questions. In this section we mention and discuss our most important findings regarding late propagation from the answers to these research questions, and focus on the possible reasons behind these findings.

Finding 1: *Type 3 clones have the highest possibility of experiencing late propagations among the three clone-types: Type 1, Type 2, and Type 3.*

From our investigations in RQ 2 we found that Type 3 clones exhibit the highest intensity of late propagations among the three clone-types (Type 1, Type 2, and Type 3). A possible reason behind why Type 3 clones exhibit a higher tendency of late propagations is that Type 3 clones are gapped clones.

Because of the existence of these gaps (i.e., non-cloned lines) consistent changing of Type 3 clones is not always as straight forward as of the other two clone-types. We suspect that higher percentage of inconsistencies as well as late propagations in Type 3 clones are caused by the gaps. However, we do not have enough supporting evidence for this. In future, we would like to investigate whether different levels of dissimilarities (i.e., different dissimilarity thresholds) in Type 3 clones have effects on the intensity of late propagations in such clones.

Finding 2: *Late propagations in Type 3 clones have the highest possibility of introducing bugs and inconsistencies to the code-base compared to the late propagations in the other two clone-types: Type 1 and Type 2.*

From our answers to the research questions RQ 3 and RQ 4 we understand that the late propagations in Type 3 clones have a higher tendency of introducing bugs and inconsistencies to a software system's code-base compared to the late propagations in each of the other two clone-types. We again suspect that the main reason behind such a scenario is the existence of gaps in Type 3 clone fragments. However, we do not have supporting evidence for this. We would like to investigate this in future.

Finding 3: *Creating block clones is more risky than creating method clones because late propagations mostly occur in block clones.*

According to our analysis in RQ 6, the clone fragments that experience late propagations are mostly block clones rather than method clones. Such a finding implies that creating block clones is more risky than creating method clones. The reason behind why a significantly higher proportion of block clones experience late propagations compared to method clones is that block clones do not have well defined boundaries as of method clones. Thus, tracking as well as consistent updating of block clones without proper tool support is intuitively much more difficult compared to method clones. Here, we should again note that an existing tool called *CloneTracker* [DR08] provides support for tracking and simultaneous editing of clone fragments. However, this tool tracks only the programmer selected clone fragments. Currently there is no tool for automatic tracking of all clone fragments in a software system. Such a tool could help us minimize late propagations in code clones considerably.

Finding 4: *Managing SPCP clones can help us minimize the occurrences of late propagations considerably.*

From our investigation regarding RQ 7 we observe that late propagations mainly occur to SPCP clones (i.e., the clones that evolve following a similarity preserving change pattern called SPCP). A previous study [MRS14b] suggests us to mainly consider SPCP clones for refactoring and tracking. As late propagation clones are mostly SPCP clones, we believe that managing of SPCP clones will help us minimize late propagations considerably. Moreover, from our findings we confirm that SPCP clones are the most important candidates from a clone management perspective. We also believe that a clone tracker with the capability of automatically detecting and tracking of SPCP clones could help programmers efficiently manage code clones by minimizing late propagations.

5 Related Work

A number of studies have already been done on clone evolution and late propagation in clones during evolution. Kim et al. [KSNM05] studied clone evolution by defining and extracting clone genealogies from two Java systems using CCFinder[5] as the clone detector. Krinke [Kri07] studied the consistent and inconsistent changes to the Type 1 clones considering the evolutions of five open source subject systems using Simian[6] clone detector. He also studied the stability of clones [Kri08] in comparison with non-cloned code. Göde et al [GH11, GK11] analyzed clone evolution and its effect on software maintenance by enhancing Krinke's study [Kri08].

In a recent study, Barbour et al. [BKZ13] investigated eight different patterns of late propagation by studying three open-source subject systems written in Java and identified two patterns that have higher likelihood of introducing inconsistencies to a code-base. They used three clone detection tools NiCad, CCFinder, and Simian in their study. However, Type 3 clones were not considered in this study. Aversano et al. [ACP07] investigated clone evolution on two subject systems to determine how clones are maintained. According to their observation 18% of the clones experienced late propagation. They show that late propagation in clones can directly be related to bugs and thus late propagation is risky. In another study Thummalapenta et al. [TCAP09] investigate late propagation in clones considering four subject systems and reported that late propagation is often related to faults and inconsistencies.

Bazrafshan [Baz12] conducted an empirical study to investigate how differently the near-miss clones evolve compared to the identical clones. He investigated the conversion of near-miss clones to identical clones, and also, identical clones to near-miss clones. According to his findings, near-miss clones should be given a higher priority than the identical clones when taking clone management decisions. Our study is different in the sense that we investigate the intensity and harmfulness of late-propagation in three clone-types (Type 1, Type 2, and Type 3) separately.

In a previous study [MRS15] we investigated and compared the bug-proneness of code clones in different clone-types. We did not investigate late propagation in that study. However, in our study presented in this paper we detect late propagation in different types of code clones, and investigate whether late propagation in code clones is related to bugs. Thus, our contributions in this study are different than in our previous study [MRS15].

We see that while there are a number of great studies, none of these focus on the intensity and harmfulness of late propagation separately in different types of clones. Also, the existing studies did not investigate the tendency of late propagation in SPCP clones (i.e., the clone fragments that evolve following a Similarity Preserving Change Pattern). In this study, we investigate these

[5] CCFinder. `http://www.ccfinder.net/ccfinderxos.html`
[6] Simian. `http://www.harukizaemon.com/simian/index.html`.

issues by answering seven research questions. We believe that our findings are important and have the potential to help us in better clone maintenance.

6 Threats to Validity

The number as well as the percentage of clone genealogies that experienced late propagation may vary because of the variation of the detection parameters of the clone detection tool (NiCad in our study). Wang et al. [WHJK13] defined this problem as the *confounding configuration choice* problem and conducted an extensive study considering six clone detectors to ameliorate the effects of the problem. However, the settings that we have used for NiCad are considered standard and with these settings NiCad can detect clones with higher precision and recall [RC09, RCK09]. Thus, the experimental results reported in this paper are of significant importance.

NiCad can detect three types of clones: Type 1 (identical), Type 2 (near-miss), and Type 3 (near-miss). Any other near-miss clone detectors [SR14] could provide us with different experimental results as well as different scenarios. However, Svajlenko and Roy [SR14] showed that NiCad is a very good clone detector for detecting all three types of code clones in comparison with the other modern clone detectors. Also, NiCad can report three types of clones (Type 1, Type 2, Type 3) separately. Thus, it helped us to investigate the intensity of late propagation on these clone-types separately.

Our research involves the detection of bug-fix commits. The way we detect such commits is similar to the technique followed by Barbour et al. [BKZ13]. Such a technique proposed by Mocus and Votta [MV00] can sometimes select a non-bug-fix commit as a bug-fix commit mistakenly. Barbour et al. [BKZ13] showed that this probability is very low. According to their investigation, the technique has an accuracy of 87% in detecting bug-fix commits. We also perform manual investigations on the bug-fix commits detected from our subject systems. As mentioned in Section 4.3, we confirmed that around 84% of these commits are true positives. Thus, we believe that our reported results regarding the bug-proneness of different types of late propagations in code clones are considerable.

In case of a Type 3 clone pair it might happen that one of the two clone fragments experienced particular changes, however, the two fragments were still considered as clones. The particular changes experienced by one fragment might later be propagated to the other fragment. Our late propagation detection mechanism cannot identify such type of late propagation where the participating clone fragments always preserve their similarity during the whole period of propagation.

Two clone fragments of a particular clone type might be regarded as clone fragments of another type after experiencing the diverging period. Bazrafshan [Baz12] previously investigated on such conversions of clone types. Xie et al. [XF13] called it clone mutation and performed an in-depth investigation regarding this. According to their investigation on three subject systems, up to 60% of the clone genealogies can experience mutation. In our research we were concerned about the late propagations in each of the three clone-types (Type 1, Type 2, and Type 3) separately. We ignored type conversions (i.e., mutations) of code clones. However, if a clone-pair is considered of different clone-types during different periods of evolution, we find the late propagations experienced by the clone-pair considering each duration separately. Thus, our experimental results regarding late propagation considering individual clone-type are not affected by the type-conversion of code clones.

The difference between the number of clone genealogies in different clone-types might be a confounding factor behind our finding regarding the comparative scenario of experiencing late propagations by different clone-types. We wanted to investigate whether this is true. Our finding is that *Type 3 clones have a higher possibility of experiencing late propagations than Type 1 and Type 2 clones*. We investigate whether this finding has been affected by the number of clone genealogies in different clone-types. We first consider the two clone-types: Type 1 and Type 3. From Table 4 we see that for five subject systems (Ctags, BRL-CAD, Carol, Jabref, and Java-ML) the number of Type 3 clone genealogies is higher than the number of Type 1 clone genealogies. For the remaining three systems (Camellia, jEdit, and Freecol), the number of Type 1 clone genealogies is higher. However, from Table 5 we see that for each of these eight systems, the percentage of clone genealogies that experienced late propagation is much higher in Type 3 case than in Type 1 case. Thus, it seems that the numbers of clone genealogies in Type 1 and Type 3 case do not impact the comparative scenario of experiencing late propagations by the code clones of these two types. Now, we make a comparison between the clone-types: Type 2 and Type 3. For each of our eight subject systems, the number of Type 2 clone genealogies is much lower compared to Type 3 (c.f., Table 4). However, for three systems (Camellia, Freecol, and Java-ML) the proportion of late propagation clone genealogies is higher in Type 2 case compared to Type 3 (c.f., Table 5). Thus, we again see that the total numbers of clone genealogies in the two clone-types (Type 2, and Type 3) do not affect the comparative scenario of experiencing late propagations by the code clones of these two types. Finally, we believe that our findings are not affected by the number of clone-genealogies in different clone-types.

A clone pair which was created just before the last revision of our investigated evolution history of a candidate system, and diverged at the last revision can converge in near future (i.e., shortly after the last revision) which is unknown to us. The earliest possible revision of convergence can be the one which will be created just after the last revision. However, as the future is unknown to us we consider this pair as a non-SPCP clone pair in our experiment. We should also note that this pair has not yet completed experiencing a late propagation according to the known evolution history. Thus, we believe that our decision about considering this pair as a non-SPCP clone pair is reasonable and such a consideration has not affected our findings.

The number of subject systems that we have used in our experiment is not sufficient to take a concrete decision regarding the possible causes of late propagation. However, we selected our subject systems focusing on the diversity of sizes (from small to large) and application domains (five different application domains) to generalize our findings. Thus, we believe that our findings are important and have the potential to minimize late propagation in clones.

7 Conclusion

In this paper, we investigate late propagation in three types of clones (Type 1, Type 2, and Type 3) separately. Through our experiment we tried to answer seven important research questions (mentioned in the Introduction) regarding the intensity, and bug-proneness of late propagation. According to our study on thousands of revisions of eight diverse subject systems written in two programming languages,

- The percentage of late propagations in Type 3 clones occurred only because of the changes in the non-matched (i.e., non-cloned) portions of the clone fragments is very low (less than one) for most of our candidate systems. However, this proportion can be considerable for some subject systems (for example our subject system jEdit). Such late propagations should be ignored when making clone management decisions. Our implemented system can automatically detect such ignorable late propagations so that we can discard these from considerations while taking clone management decisions.
- The intensity of late propagation in Type 3 clones is higher compared to the other two clone-types (Type 1, and Type 2).
- More importantly, Type 3 clones have higher possibilities of experiencing buggy late propagations than the clone fragments in the other two types.
- Most of the clone fragments that experience late propagations are block clones. It seems that the creation of block clones is more risky than the creation of method clones.
- Refactoring and tracking of SPCP-clones can possibly help us minimize the future occurrences of late propagations considerably.

As a future work, we plan to investigate whether programming languages as well as application domains of the subject systems can bias the intensity of late propagation. Considering Type 3 clones, we plan to investigate different late propagation patterns, their frequencies and effects on software maintenance.

References

[ACP07] L. Aversano, L. Cerulo, M. D. Penta. How Clones Are Maintained: An Empirical Study. In *CSMR*. Pp. 81–90. IEEE Computer Society, 2007.

[Baz12] S. Bazrafshan. Evolution of Near-Miss Clones. In *SCAM*. Pp. 74 – 83. 2012.

[BKZ13] L. Barbour, F. Khomh, Y. Zou. An empirical study of faults in late propagation clone genealogies. *Software: Evolution and Process* 25:1139 – 1165, 2013.

[CR11] J. R. Cordy, C. K. Roy. The NiCad Clone Detector. In *Tool Demo Track, ICPC*. Pp. 219 – 220. 2011.

[DLL09] M. D'Ambros, M. Lanza, M. Lungu. Visualizing co-change information with the evolution radar. *IEEE Transactions on Software Engineering* 35:720 – 735, 2009.

[DR07] E. Duala-Ekoko, M. P. Robillard. Tracking Code Clones in Evolving Software. In *ICSE*. Pp. 158 – 167. 2007.

[DR08] E. Duala-Ekoko, M. P. Robillard. CloneTracker: Tool Support for Code Clone Management. In *ICSE*. Pp. 843 – 846. 2008.

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

[effb] Effect Size Calculation for MannWhitneyWilcoxon Test.
`http://www.let.rug.nl/~heeringa/statistics/stat03_2013/lect09.pdf`

[GH11] N. Göde, J. Harder. Clone Stability. In *CSMR*. Pp. 65–74. 2011.

[GK11] N. Göde, R. Koschke. Frequency and risks of changes to clones. In *ICSE*. Pp. 311 – 320. 2011.

[KG08] C. Kapser, M. W. Godfrey. "Cloning considered harmful" considered harmful: patterns of cloning in software. *Empirical Software Engineering* 13:645 – 692, 2008.

[Kri07] J. Krinke. A study of consistent and inconsistent changes to code clones. In *WCRE*. Pp. 170 – 178. 2007.

[Kri08] J. Krinke. Is cloned code more stable than non-cloned code? In *SCAM*. Pp. 57 – 66. 2008.

[KSNM05] M. Kim, V. Sazawal, D. Notkin, G. C. Murphy. An empirical study of code clone genealogies. In *ESEC-FSE*. Pp. 187 – 196. 2005.

[LW08] A. Lozano, M. Wermelinger. Assessing the effect of clones on changeability. In *ICSM*. Pp. 227 – 236. 2008.

[LW10] A. Lozano, M. Wermelinger. Tracking clones' imprint. In *IWSC*. Pp. 65 – 72. 2010.

[MRR$^+$12] 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. In *ACM SAC*. Pp. 1227–1234. ACM, 2012.

[MRS12a] M. Mondal, C. K. Roy, K. A. Schneider. Connectivity of co-changed method groups: a case study on open source systems. In *CASCON*. Pp. 205 – 219. 2012.

[MRS12b] M. Mondal, C. K. Roy, K. A. Schneider. Dispersion of changes in cloned and non-cloned code. In *IWSC*. Pp. 29 – 35. 2012.

[MRS12c] M. Mondal, C. K. Roy, K. A. Schneider. An Empirical Study on Clone Stability. *ACM SIGAPP Applied Computing Review* 12(3):20–36, 2012.

[MRS14a] M. Mondal, C. K. Roy, K. A. Schneider. Automatic Identification of Important Clones for Refactoring and Tracking. In *SCAM*. P. 10pp. (to appear). 2014.

[MRS14b] M. Mondal, C. K. Roy, K. A. Schneider. Automatic Ranking of Clones for Refactoring through Mining Association Rules. In *CSMR-WCRE*. Pp. 114 – 123. 2014.

[MRS14c] M. Mondal, C. K. Roy, K. A. Schneider. An insight into the dispersion of changes in cloned and non-cloned code: A genealogy based empirical study. *Science of Computer Programming* 95:445 – 468, 2014.

[MRS14d] M. Mondal, C. K. Roy, K. A. Schneider. Late Propagation in Near-Miss Clones: An Empirical Study. In *IWSC*. P. 17pp. 2014.

[MRS15] M. Mondal, C. K. Roy, K. A. Schneider. A Comparative Study on the Bug-Proneness of Different Types of Code Clones. In *ICSME*. Pp. 91 – 100. 2015.

[MV00] A. Mockus, L. G. Votta. Identifying Reasons for Software Changes using Historic Databases. In *ICSM*. Pp. 120 – 130. 2000.

[mwwa] MannWhitneyWilcoxon Test.
`http://en.wikipedia.org/wiki/Mann%E2%80%93Whitney_U_test`

[mwwb] MannWhitneyWilcoxon Test Online.
`http://elegans.som.vcu.edu/~leon/stats/utest.cgi`

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

[RC08] C. K. Roy, J. R. Cordy. NICAD: Accurate Detection of Near-Miss Intentional Clones Using Flexible Pretty-Printing and Code Normalization. In *ICPC*. Pp. 172–181. IEEE Computer Society, 2008.

[RC09] C. K. Roy, J. R. Cordy. A mutation / injection-based automatic framework for evaluating code clone detection tools. In *Mutation*. Pp. 157–166. 2009.

[RCK09] 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* 74:470 – 495, 2009.

[Roy09] C. K. Roy. Detection and analysis of near-miss software clones. In *ICSM*. Pp. 447–450. 2009.

[SR14] J. Svajlenko, C. K. Roy. Evaluating Modern Clone Detection Tools. In *ICSME*. Pp. 321 – 330. 2014.

[TCAP09] S. Thummalapenta, L. Cerulo, L. Aversano, M. D. Penta. An empirical study on the maintenance of source code clones. *Empirical Software Engineering* 15:1 – 34, 2009.

[VPV10] A. Vanya, R. Premraj, H. V. Vliet. Interactive Exploration of Co-evolving Software Entities. In *CSMR*. Pp. 260 – 263. 2010.

[WHJK13] T. Wang, M. Harman, Y. Jia, J. Krinke. Searching for Better Configurations: A Rigorous Approach to Clone Evaluation. In *ESEC/SIGSOFT FSE*. Pp. 455 – 465. 2013.

[XF13] S. Xie, Y. Z. F. Khomh. An Empirical Study of the Fault-Proneness of Clone Mutation and Clone Migration. In *MSR*. Pp. 149 – 158. 2013.

Biographies of the Authors

Manishankar Mondal

Manishankar Mondal is a graduate student in the Department of Computer Science of the University of Saskatchewan, Canada under the supervision of Dr. Chanchal Roy and Dr. Kevin Schneider. He is a lecturer at Khulna University, Bangladesh and currently on leave for pursuing his higher studies. He received the Best Paper Award from the 27th Symposium On Applied Computing (ACM SAC 2012) in the Software Engineering Track. His research interests are software maintenance and evolution including clone detection and analysis, program analysis, empirical software engineering and mining software engineering.

Chanchal K. Roy

Chanchal Roy is an associate professor of Software Engineering/Computer Science at the University of Saskatchewan, Canada. While he has been working on a broad range of topics in Computer Science, his chief research interest is Software Engineering. In particular, he is interested in software maintenance and evolution, including clone detection and analysis, program analysis, reverse engineering, empirical software engineering and mining software repositories. He served or has been serving in the organizing and/or program committee of major software engineering conferences (e.g., ICSM, WCRE, ICPC, SCAM, ICSE-tool, CASCON, and IWSC). He has been a reviewer of major Computer Science journals including IEEE Transactions on Software Engineering, International Journal of Software Maintenance and Evolution, Science of Computer Programming, Journal of Information and Software Technology and so on. He received his Ph.D. at Queen's University, advised by James R. Cordy, in August 2009.

Kevin A. Schneider

Kevin Schneider is a Professor of Computer Science, Special Advisor ICT Research and Director of the Software Engineering Lab at the University of Saskatchewan. Dr. Schneider has previously been Department Head (Computer Science), Vice-Dean (Science) and Acting Chief Information Officer and Associate Vice-President Information and Communications Technology.
Before joining the University of Saskatchewan, Dr. Schneider was CEO and President of Legasys Corp., a software research and development company specializing in design recovery and automated software engineering. His research investigates models, notations and techniques that are designed to assist software project teams develop and evolve large, interactive and usable systems. He is particularly interested in approaches that encourage team creativity and collaboration.

Photos of the Authors

Manishankar Mondal

Chanchal K. Roy

Kevin A. Schneider