A Comparative Study of Software Bugs in Clone and Non-Clone Code

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

Abstract—Code cloning is a recurrent operation in everyday software development. Whether it is a good or bad practice is an ongoing debate among researchers and developers for the last few decades. In this paper, we conduct a comparative study on bug-proneness in clone code and non-clone code by analyzing commit logs. According to our inspection on thousands of revisions of seven diverse subject systems, the percentage of changed files due to bug-fix commits is significantly higher in clone code compared with non-clone code. We perform a Mann-Whitney-Wilcoxon (MWW) test to show the statistical significance of our findings. Finally, the possibility of severe bugs occurring is higher in clone code than in non-clone code. Bug-fixing changes affecting clone code should be considered more carefully. According to our findings, clone code appears to be more bug-prone than non-clone code.

I. INTRODUCTION

If two or more code fragments in a software system’s code-base are exactly or nearly similar to one another we call them code clones [1], [2]. A group of similar code fragments forms a clone class. Code clones are mainly created because of the frequent copy/paste activities of programmers during software development and maintenance [1][1]

A significant number of studies [3]–[21] have been conducted on discovering the impact of cloning on software maintenance. While a number of studies [3], [6], [7], [9]–[12] have revealed some positive sides of code cloning, there is strong empirical evidence [4], [5], [8], [13]–[15], [17], [19]–[21] of negative impacts of code clones too. These negative impacts include higher instability [15], late propagation [4], and unintentional inconsistencies [5]. Existing studies [4], [22] show that code clones are related to bugs in the code-base.

Several studies have showed that bugs have a great effect on code in software systems. Our previous study [23] provides evidence of bug-replication in clone code. Also, Sajnani et al. [24] showed that cloned code has less problematic bug patterns than non-cloned code. They have used bugs reported by FindBugs from just one snapshot of the last revision of the system. Whereas we consider bugs reported during the evolution of a software system through thousands of commits. We conduct our analysis considering both exact (Type 1) and near-miss clones (Type 2 and Type 3 clones) [1], [2]. The clone-types are defined below.

Type 1 Clones. If two or more code fragments in a particular code-base are exactly the same disregarding the
TABLE I
RESEARCH QUESTIONS

<table>
<thead>
<tr>
<th>SL</th>
<th>Research Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>RQ 1</td>
<td>What percentage of files get affected because of clone and non-clone bug-fix commits?</td>
</tr>
<tr>
<td>RQ 2</td>
<td>How often do bug-fix changes occur to the clone and non-clone code?</td>
</tr>
<tr>
<td>RQ 3</td>
<td>Is there any difference between the severity of the bugs occurring in clone and non-clone code?</td>
</tr>
</tbody>
</table>

TABLE II
SUBJECT SYSTEMS

<table>
<thead>
<tr>
<th>Systems</th>
<th>Lang.</th>
<th>Domains</th>
<th>LLR</th>
<th>Revisions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctags</td>
<td>C</td>
<td>Code Def. Generator</td>
<td>33,270</td>
<td>774</td>
</tr>
<tr>
<td>Camellia</td>
<td>C</td>
<td>Image Processing Library</td>
<td>89,063</td>
<td>170</td>
</tr>
<tr>
<td>BrJCAD</td>
<td>C</td>
<td>Solid Modeling CAD</td>
<td>39,309</td>
<td>735</td>
</tr>
<tr>
<td>jEdit</td>
<td>Java</td>
<td>Text Editor</td>
<td>191,804</td>
<td>4000</td>
</tr>
<tr>
<td>Freecol</td>
<td>Java</td>
<td>Game</td>
<td>91,626</td>
<td>1950</td>
</tr>
<tr>
<td>Carol</td>
<td>Java</td>
<td>Game</td>
<td>25,091</td>
<td>1700</td>
</tr>
<tr>
<td>JabRef</td>
<td>Java</td>
<td>Reference Management</td>
<td>45,515</td>
<td>1545</td>
</tr>
</tbody>
</table>

LLR = LOC in the Last Revision

TABLE III
NiCad SETTINGS FOR THREE TYPES OF CLONES

<table>
<thead>
<tr>
<th>Clone Types</th>
<th>Identifier Renaming</th>
<th>Dissimilarity Threshold</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type 1</td>
<td>none</td>
<td>0%</td>
</tr>
<tr>
<td>Type 2</td>
<td>blindrename</td>
<td>0%</td>
</tr>
<tr>
<td>Type 3</td>
<td>blindrename</td>
<td>20%</td>
</tr>
</tbody>
</table>

We use NiCad [25] for detecting block clones including both exact (Type 1) and near-miss (Type 2, Type 3) clones of a minimum size of 10 LOC with 20% dissimilarity threshold and blind renaming of identifiers. NiCad settings for detecting three clone-types (Type 1, Type 2, and Type 3) are shown in Table III. For different settings of a clone detector the clone detection results can be different and thus, the findings on bugs in code clones can also be different. Hence, selection of appropriate settings (i.e., detection parameters) is important. We used the mentioned settings in our research, because [30] Svaženko and Roy show that these settings provide us with better clone detection results in terms of both precision and recall. Moreover, code clones with a minimum size of 10 LOC are more appropriate from maintenance perspectives [1], [31], [32]. Before using the NiCad outputs of Type 2 and Type 3 cases, we 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 processed NiCad clone detection results in the mentioned ways because we wanted to investigate bug in three types of clones separately.

B. Bug-proneness Detection Technique

For each subject system, we first retrieve the commit messages by applying the ‘SVN log’ command. A commit message describes the purpose of the corresponding commit operation. We automatically infer the commit messages using the heuristic proposed by Mockus and Votta [33] in order to identify those commits that occurred for the purpose of fixing bugs. Then we identify which of these bug-fix commits make changes to clone fragments. If one or more clone fragments are modified in a particular bug-fix commit, then it is an implication that the modification of those clone fragment(s) was necessary for fixing the corresponding bug. In other words, the clone fragment(s) are related to the bug. In this way we examine the commit operations of a candidate system, analyze the commit messages to retrieve the bug-fix commits, and identify those clone fragments that are related to the bug-fix. We also determine the number of changes that occurred to such a clone fragment in a bug-fix commit using the UNIX diff command.

The procedure that we follow to detect the bug-fix commits was also previously followed by Barbour et al. [4]. Barbour et al. [4] detected bug-fix commits in order to investigate whether late propagation in clones is related to bugs. They at
first identified the occurrences of late propagations and then analyzed whether the clone fragments that experienced late propagations are related to a bug-fix. In our study we detect bug-fix commits in the same way as they detected, however, our study is different in the sense that we investigate the bugs of different types of code clones. Also, Barbour et al. [4] did not investigate the most important clone type, the Type 3. Generally, the number of Type 3 clones in a system is the highest among the three clone-types. We consider Type 3 clones in our bug-fix study.

IV. EXPERIMENTAL RESULTS AND ANALYSIS

We mention our three research questions in Table I. In this section we present our experimental results and analyze them to find the answers to our research questions.

A. Answering the first research question (RQ 1)

RQ 1: What percentage of files get affected because of clone and non-clone bug-fix commits?

Motivation. It is important to know the percentage of affected files due to bug-fixing commits and compare between clone and non-clone code. More affected files means more changes in the system. More attention is needed when more changes occur. Knowing the information we can emphasize on which type of code (clone or non-clone) are affecting the system more.

Methodology. To answer this research question we automatically count the total number of files that contain clone code in three different types of clones (Type 1, Type 2 and Type 3) and total number of files containing non-clone code. Also, we detect the total number of files that contain changed clone in three different clone types and the total number of files containing changed non-clone code. Then we calculate their percentages for individual clone types.

FC: This is the total number of files that have clone code. Columns with the heading FC in Table IV represents the value.

FCC: This is the total number of files that contain changed clone code. This value is given in columns with the heading FCC in Table IV.

PFCC: To find out the percentage of the number of files that have changes in source code. From these files we identify the total number of files that have changes in clone code. The rest of the files made changes to non-clone code due to bug-fix commits.

FNC: This is the number of total files that have non-clone code. Columns with the heading FNC in Table V show the values.

FCNC: This is the number of total files which contain changes in non-clone code. To find out this file number we consider those files which contain changed non-clone code. We also check those files which have changed clone code in addition with non-clone code. All the columns of Table V with the heading FCNC show this value.

PFCNC: To calculate percentage of the number of files containing changed non-clone code we use following equation. All the columns with the heading PFCNC in Table V represent this value. This is shown in equation 3.

\[
\text{PFCNC} = \frac{100 \times \text{FCNC}}{\text{FNC}}
\]  

(3)

OPFCNC: We calculate the overall percentage of files containing changed non-clone code using following equation.

\[
\text{OPFCNC} = \frac{100 \times \sum \text{all systems FCNC}_{Ti}}{\sum \text{all systems FNC}}
\]  

(4)

PFCNCs of different clone types for each subject system are shown in Figure 1. Here, we can see that Camellia has
TABLE IV

<table>
<thead>
<tr>
<th>Subject Systems</th>
<th>Type 1</th>
<th>Type 2</th>
<th>Type 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FC</td>
<td>FCC</td>
<td>PFCC</td>
</tr>
<tr>
<td>Ctags</td>
<td>12</td>
<td>4</td>
<td>33.33%</td>
</tr>
<tr>
<td>Camellia</td>
<td>11</td>
<td>2</td>
<td>18.18%</td>
</tr>
<tr>
<td>Brlcad</td>
<td>33</td>
<td>3</td>
<td>9.09%</td>
</tr>
<tr>
<td>jEdit</td>
<td>21338</td>
<td>117</td>
<td>0.54%</td>
</tr>
<tr>
<td>Freecol</td>
<td>259</td>
<td>13</td>
<td>5.01%</td>
</tr>
<tr>
<td>Carol</td>
<td>121</td>
<td>14</td>
<td>11.57%</td>
</tr>
<tr>
<td>Jabref</td>
<td>97</td>
<td>8</td>
<td>8.24%</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>4</td>
<td>33.33%</td>
</tr>
<tr>
<td>Camellia</td>
<td>25</td>
<td>4</td>
<td>16%</td>
</tr>
<tr>
<td>Brlcad</td>
<td>5</td>
<td>1</td>
<td>20%</td>
</tr>
<tr>
<td>jEdit</td>
<td>695</td>
<td>14</td>
<td>2.01%</td>
</tr>
<tr>
<td>Freecol</td>
<td>178</td>
<td>11</td>
<td>6.17%</td>
</tr>
<tr>
<td>Carol</td>
<td>138</td>
<td>14</td>
<td>10.14%</td>
</tr>
<tr>
<td>Jabref</td>
<td>81</td>
<td>7</td>
<td>8.64%</td>
</tr>
<tr>
<td>Overall</td>
<td></td>
<td></td>
<td>10.25%</td>
</tr>
</tbody>
</table>

\[ FC = \text{Number of Files that have Clone code} \]
\[ FCC = \text{Number of Files that have Changed Clone code} \]
\[ PFCC = \text{Percentage of the Number of Files that have Changed Clone code} \]

The highest percentage than rest of the subject systems. On the other hand, jEdit has the lowest percentage among all subject systems. Table V shows FNC, FCNC and PFCNC for all subject systems. We observe that percentage of changes due to bug-fix commits is higher in clone code than non-clone code. This result was expected because total number of files containing non-clone code is much higher (almost three times) than clone code in every subject system.

The overall percentages (OPFCC) of files that have changes in Type 1 (12.28%) and Type 2 (8.99%) clones have more changes in files than Type 3 (5.63%) clones for bug-fix commits. Moreover, the percentage of files that have changes in clone code is higher than the percentage (OPFCNC) of files that have changes in non-clone code (1.52%).

**Mann-Whitney-Wilcoxon (MWW) tests for RQ 1.** We are interested to know whether the percentages of three clone types is significantly higher than non-clone code. First, we perform Mann-Whitney-Wilcoxon (MWW) test [34] with percentages of Type 1 clone and non-clone code. We consider the significance level is 5% for this test. According to the data critical U is 13. If the p-value is less than 0.05 and U value is less than 13 then the result is significant. Our result shows that percentage of Type 1 clone code is significantly higher than non-clone code. For two-tailed test we find the p-value of 0.011719 which is much lower than 0.05. In the same way we perform the test for percentages of Type 2 clone code and Type 3 clone code with the non-clone code. We find the p-value of 0.015714 and 0.004574 for Type 2 clone and Type 3 clone respectively. Both percentages of Type 2 clone and Type 3 clone code are significantly higher than non-clone code. Thus, we can say that percentages of all three types of clones are significantly higher than non-clone code. We list our MWW test results in Table VI.

**Answer to RQ 1.** According to our experimental results percentage of number of file changes in bug-fix commits is higher in clone code than non-clone code. Also, in terms of overall percentage of files Type 1 and Type 2 code clones (12.28% and 8.99%) have higher percentage than Type 3 code clone (5.63%).

We observe that percentages of files containing changes
due to bug-fix commits is high in clone code than non-clone code. However, we still do not know what percentage of clone and non-clone code get changed during bug-fix commits. Intuitively, percentages of bug-fix changes should be higher in clone code than non-clone code. To understand this we investigate our next research question.

B. Answering the second research question (RQ 2)

**RQ 2: How often do bug-fix changes occur to the clone and non-clone code?**

**Motivation.** Though we have the answer of RQ 1 and hence we know the percentage of files changed in clone and non-clone code but still we are not sure how much these changes are influencing the system. It is important to know the frequency of the bug-fix changes in both clone and non-clone code. From comparison between them we can understand the impact of bug-fix changes. Intuitively, more importance should be given to the more frequent one. This will help us to manage clone code.

**Methodology.** We know the total number of commits of each subject system. As discussed in Section III-A, we report the total number of commits that have changes in clone fragments. To answer the RQ 2 we automatically count the total number of bug-fix commits that contain changes in clone code. First, we find out the total number of bug-fix commits as described in Section III-B. We automatically count the number of total bug-fix commits which have changed clone code. We deduct this number from the total number of bug-fix commits and found the total number of bug-fix commits which have changed non-clone code. In the following way we calculate the occurrences of bug-fix changes in clone and non-clone code.

CC: This is the total Number of Commits that made changes to Clone code. All the columns of Table VII with the heading CC show this value.

BCC: This is the total Number of Bug-fix Commits that made changes to Clone code. Columns with the heading BCC in Table VII show the values.

PBCC: Percentage of the Bug-fix Commits that made changes to Clone code. We calculate this for each subject systems and for three different types of clone i.e. Type 1, Type 2 and Type 3 clone code. All the columns with the heading PBCC in Table VII represent this value.

We use the following equation for calculating the percentage.

$$PBCC = \frac{100 \times BCC}{CC}$$ (5)

We calculate the overall percentage of the bug-fix commits containing clone code using the following equation.

$$OPBCC_{T_i} = \frac{100 \times \sum_{all\ systems} BCC_{T_i}}{\sum_{all\ systems} CC_{T_i}}$$ (6)

Here, $OPBCC_{T_i}$ is the overall percentage of the clone code found in the bug-fix commits with respect to $T_i$ type of clones ($i=1, 2, 3$). Table VII shows the value of CC, BCC and PBCC for seven subject systems and each type of clone individually. Figure 2 describes the percentage PBCCs of all subject systems along with the overall percentage OPBCC. We can see that jEdit has the highest percentage (over 40%) and Brlcad has the lowest percentage (less than 15%). However, in overall percentage Type 3 code clone has the higher percentage than Type 1 and Type 2 code clone.

We first identify the list of commits that made changes to the source code. From these commits we detect which commits made changes to clone code. The remaining commits in the list made changes to non-clone code. In the same way we deduct the total number of bug-fix commits containing changes in clone code from the total number of bug-fix commits to find the number of changes in non-clone code bug-fix commits. Applying these findings we answer RQ 2.

CNC: This is the total Number of Commits that made changes to Non-clone code. This value is given in columns with the heading CNC in Table VIII.

BCNC: This is the total Number of Bug-fix Commits that made changes to Non-clone code. Columns with the heading BCNC in Table VIII represent the value.

PBCNC: Percentage of the Bug-fix Commits that made changes to Non-clone code. In Table VIII columns with the heading PBCNC show this value.

Likewise equation 5 to compute the percentages we use equation 7.

$$PBCNC = \frac{100 \times BCNC}{CNC}$$ (7)

To see the overall percentage of the number of bug-fix commits that is related to non-clone code we use following equation which is similar to the equation 6.

$$OPBNCNC = \frac{100 \times \sum_{all\ systems} BCNC}{\sum_{all\ systems} CNC}$$ (8)

Here, $OPBNCNC$ is the overall percentage of the bug-fix commits that contain non-clone code. The CNC, BCNC and PBCNC for all subject systems are shown in Table VIII. Here, most of the percentage ranges from 15% to 35% (only the percentage of Freecol has more than 45%). Figure 2 depicts the percentage PBCNCs and overall percentage OPBNCNC of
TABLE VII
NUMBER OF BUG-FIX COMMITS AFFECTING CLONE CODE

<table>
<thead>
<tr>
<th>Subject Systems</th>
<th>Type 1</th>
<th>Type 2</th>
<th>Type 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CC</td>
<td>BCC</td>
<td>PBCC</td>
</tr>
<tr>
<td>Ctags</td>
<td>14</td>
<td>3</td>
<td>21.42%</td>
</tr>
<tr>
<td>Camellia</td>
<td>8</td>
<td>1</td>
<td>12.5%</td>
</tr>
<tr>
<td>Brlcad</td>
<td>32</td>
<td>2</td>
<td>6.25%</td>
</tr>
<tr>
<td>jEdit</td>
<td>92</td>
<td>37</td>
<td>40.21%</td>
</tr>
<tr>
<td>Freecol</td>
<td>35</td>
<td>7</td>
<td>20%</td>
</tr>
<tr>
<td>Carol</td>
<td>41</td>
<td>8</td>
<td>19.51%</td>
</tr>
<tr>
<td>Jabref</td>
<td>48</td>
<td>6</td>
<td>12.5%</td>
</tr>
</tbody>
</table>

CC = Number of Commits affecting Clone code
BCC = Number of Bug-fix Commits affecting Clone code
PBCC = Percentage of Commits that were applied for fixing Bugs in Clone code

seven subject systems. Here, we observe that percentage of the bug-fix commits containing changed non-clone code is highest in Freecol system and lowest in Brlcad. Overall percentage is near 30% which is higher than clone code. Though the bug-fix changes occur more in non-clone code than clone code it is not noteworthy since the difference is not that much high.

The overall percentage (OPBCC) of Type 3 (22.32%) clone is higher than Type 1 (18.91%) and Type 2 (21.56%) clone. Obviously, Type 1 clone (has exact same code) and Type2 clone (has renaming of identifiers or changing data type of identifiers) have less changes than Type 3 clone code (has addition, deletion or modification of code). We believe for this reason Type 3 clone code is affected more than Type 1 and Type 2 clone code. Also, the overall percentage (OPBCNC) for non-clone code (27.13%) is higher than clone code.

**Mann-Whitney-Wilcoxon (MWW) tests for RQ 2.** We perform MWW test [34] to understand whether the difference between the percentages of clone and non-clone code is significant. Percentage of each type of clone is individually tested with non-clone code. We found the p-value of 0.092892, 0.207578 and 0.344562 for Type 1, Type 2 and Type 3 clone respectively. Calculated U value for Type 1, Type 2 and Type 3 clone code is 16, 20 and 23 respectively. Here, every p-value is greater than 0.05 (significance level is 5%) and every U value is greater than 13 (critical U value is 13). This indicates the result is insignificant. Insignificant result denotes percentage of bug-fix commits having changed non-clone code is slightly higher than clone code. Table IX describes the p-value and U value for all three types of clone code.

**Answer to RQ 2.** Comparing the overall percentage of bug-fix commits containing clone and non-clone code we found that frequency of bug-fix commits is slightly higher in non-clone code (27.13%) than clone code (20.93%). Also, Type 3 clone code (22.32%) has the highest percentage of changes than Type 1 and 2 clone code (18.91% and 21.56%) due to bug-fix commits.

We observe that bug-fix commits occur in non-clone code more often than clone code by 6.2%. From MWW test we find that the difference between percentages of clone and non-clone code is insignificant.
C. Answering the third research question (RQ 3)

**RQ 3:** Is there any difference between the severity of the bugs occurring in clone and non-clone code?

**Motivation.** In every single commit there is a message or comment written by the programmer which describes about the changes that they made from the previous commit. In case of bug-fix commits these messages describe about the bug that occur in the code base of the system. By reading a bug-fix commit message we can understand whether a bug is severe or not. This message is helpful for debugging and understanding the scenario of the situation. To understand the severity of bugs and compare between clone and non-clone code we automatically process the bug-fix commit messages followed by a manually inspection. It is important to give priority to more severe bugs while fixing them.

**Methodology.** We automatically perform a heuristic search proposed by Lamkanfi et al. [35] in the bug-fix commit messages to identify severe bugs. Then we manually investigate the results for validation. We consider five subject systems (Ctags, Camellia, Brlcad, jEdit, Freecol) for this experiment. For the non-clone bug-fix commits we choose some random bug-fix commits which does not contain clone code. It is not feasible to check all the non-clone bug-fix commits by manual inspection. Hence, we keep the total number of non-clone bug-fix commits equals to the total number of clone bug-fix commits to maintain the data impartiality.

Lamkanfi et al. [35] suggested most significant terms for different components indicating severe and non-severe bugs. For example, ‘fault’, ‘hang’, ‘freeze’, ‘deadlock’ etc. represent severe problems of the system. The words ‘favicon’, ‘deprec’, ‘mnemon’, ‘outbox’ etc. represent non-severe problems of the system. We take these terms as keywords to decide the severity of the bug. Though there are different levels of bug severity we consider only two categories for the simplicity. That is whether the bug is severe or not i.e. ‘TRUE’ or ‘FALSE’. Here, ‘TRUE’ means the bug is sever (i.e. bug-fix commit messages containing the terms which represent severity) and ‘FALSE’ means the bug is non-sever (i.e. bug-fix commit messages containing the terms which represent non-severity). This is important for bug triaging process since severe bugs need more care and prompt fixing.

We calculate the percentage of severe bugs in bug-fix commits for both clone and non-clone code. We observe that the existence of severe bugs in Camellia (both clone and non-clone), Brlcad (clone) and jEdit (non-clone) systems is zero (0%). We also observe that Freecol has highest percentage (85.71% for clone and 62.5% for non-clone code) of severe bugs in both clone and non-clone bug-fix commits compared to other subject systems. Percentages of severe bugs in rest of the subject systems are range from 50% to 67%. Ctags (non-clone) and Brlcad (non-clone) have the lowest percentage (50%) of severe bugs. Overall, clone code has higher tendency of having severe bugs than non-clone code. The difference between the overall percentages of severe bugs of clone and non-clone code bug-fix commits is 17.46% which is highly significant. This findings imply that more importance should be given on clone code while fixing bugs for better software maintenance.

**Answer to RQ 3.** After careful inspection of each commit messages of the bug-fix commits for both clone and non-clone code we found that clone code bug-fix commits has higher percentage of severe bugs (overall percentage 71.63%) than the non-clone code (overall percentage 54.17%). This proves that occurrence of severe bugs is higher in clone code compared with non-clone code.

We observe that severity of bugs is higher in clone code than non-clone code. Though, we find that some of the bug-fix commit messages are very short and it is not enough to describe the severity of the bug. Considering this constraint of the messages in bug-fixing commits the result may vary in different cases. However, severe bugs should have the highest priority in software maintenance.

V. RELATED WORK

Shajnani et al. [24] performed a comparative study between clone and non-clone code for different bug patterns. They used bug reports generated by a tool, i.e., FindBugs whereas we worked on real bug reports that were reported by developers. In our previous study [23] it has been shown that cloning is responsible for replicating bugs. However, it does not show any comparison with non-clone code in that study.

Bug-proneness of code clones has been investigated by a number of existing studies. Li and Ernst [19] performed an empirical study on the bug-proneness of clones by investigating four software systems and developed a tool called CBCD on the basis of their findings. CBCD can detect clones of a given piece of buggy code. Li et al. [36] developed a tool called CP-Miner which is capable of detecting bugs related to inconsistencies in copy-paste activities. Steidl and Göde [20] investigated finding instances of incompletely fixed bugs in near-miss code clones by investigating a broad range of features of such clones involving machine learning. Göde and Koschke [5] investigated the occurrences of unintentional inconsistencies to the code clones of three mature software systems and found that around 14.8% of all changes that occurred to the code clones were unintentionally inconsistent.

None of the studies discussed above investigated bug-fix commits in code clones and non-clone code simultaneously. Mondal et al. [22] investigated bug-proneness of code clones. While the primary target of that study was to compare the bug-proneness of three clone-types, our target is to compare the bug-proneness of clone and non-clone code. Mondal et al. [22] did not investigate the bug-proneness of non-clone code in their study. Focusing on this we perform an in-depth investigation on bug’s impacts in code clones and non-clones in this research. Our experimental results are promising and provide useful implications for better understanding of the bug-proneness of clone and non-clone code.
VI. Threats to Validity

We used the NiCad clone detector [25] for detecting clones. While all clone detection tools suffer from the confusing configuration choice problem [37] and might give different results for different settings of the tools, the setting that we used for NiCad for this experiment are considered standard [38] and with these settings NiCad can detect clones with high precision and recall [28]–[30]. Thus, we believe that our findings on the bug-proneness of clone codes are of significant importance.

Our research involves the detection of bug-fix commits. The way we detect such commits is similar to the technique proposed by Mocus and Votta [33] and also used by Barbour et al. [39]. The technique proposed by Mocus and Votta [33] can sometimes select a non-bug-fix commit as a bug-fix commit mistakenly. However, Barbour et al. [39] showed that this probability is very low. According to their investigation, the technique has an accuracy of 87% in detecting bug-fix commits.

VII. Conclusion

In this paper we conduct an in-depth comparative study of software bugs in both clone and non-clone code. For clone code we also consider three major types of clones, Type 1, Type 2 and Type 3. We investigated thousands of revisions of seven diverse subject systems. We also investigated bug-fix commit messages to measure the frequency of severe bugs in clone and non-clone code. From our examination, changes to files due to bug-fix commits is higher for clone code than for non-clone code. Additionally, changes to files due to bug-fix commits happens more in Type 1 and Type 2 code clones than in Type 3 code clones. In addition, percentage of severe bugs is higher in clone code than non-clone code bug-fix commits. We believe that our findings on bug-fix commits are valuable for better understanding of clone management such as ranking of clone codes and software maintenance.

References