On the Relationships between Stability and Bug-proneness of Code Clones: An Empirical Study

Md Saidur Rahman, Chanchal K. Roy
University of Saskatchewan, Canada
{saeed.cs, chanchal.roy}@usask.ca

***Abstract*—Exact or similar copies of code fragments in a code base are known as code clones. Code clones are considered as one of the serious code smells. Stability is a widely investigated perspective of assessing the impacts of clones on software systems. A number of existing studies show that clones are often less stable than non-cloned code. This suggests that clones change more frequently than non-cloned code and thus may require comparatively more maintenance efforts. Again, frequent changes to clones may increase the likelihood of missing change propagation to the co-change candidates leading to inconsistencies or bugs. However, none of the existing studies investigate whether stability of clones is related to the bug-proneness. In this paper, we present an empirical study that analyzes the relationships between stability and bug-proneness of clones. We identify bug-fix commits by analyzing the commit messages from software repositories. We then identify the clones those are changed in the bug-fix commits as bug-prone clones. We then compare the stability of buggy and non-buggy clones considering the fine-grained syntactic change types and their significance.**

Our experimental results based on five open-source Java systems of different size and application domains show that (1) stability and bug-proneness of code clones are related and this relationship is statistically significant, (2) for both exact (Type 1) and near-miss (Type 2 and Type 3) clones, buggy clones tend to have higher frequency of changes than non-buggy clones, (3) the bug-proneness of Type 2 and Type 3 clones tend to be strongly related with their stability compared to Type 1 clones, and (4) the relation between the stability and the bug-proneness of clones with respect to fine-grained change types is likely to be influenced by the changes of low to medium significance. We believe that our findings are important and potentially useful in identifying and prioritizing candidate clones for management.

***Index Terms*—Code Clones, Clone Stability, Bug-proneness of Clones**

I. Introduction

Code reuse by copying a code fragment and pasting it with or without modification results in duplicate copies of exact or similar code fragments in the code base. These exact or similar copies of code fragments are known as code clones. As clones constitute a significant proportion (between 7% and 23% [1]) of code bases of software systems, many studies investigated the impact of clones on software systems. A number of studies [2], [3], [4], [5], [6] show that clones are not harmful and instead clones can be beneficial for software systems [7]. Although clones have some obvious benefits like increasing productivity due to faster software development by code reuse, a number of studies [8], [9], [10], [11], [12], [13], [14], [15], [16] report strong empirical evidences concluding that clones have negative impacts on software systems. Thus, code clones are considered as one of the serious code smells.

One well known claim against code clones is that code clones may introduce bugs or inconsistencies [8] in the software systems if they are not changed consistently during software evolution. Moreover, if a code fragment with unidentified bugs is cloned, hidden bugs may also propagate to the new cloned fragments [14]. Given the important concerns related to clones, there are many studies on bug-proneness of clones. Some studies focused on the identification and localization of bugs [13], [14], [17]. Some studies investigated the comparative bug-proneness of different types of clones [18], [19], [20] while some studies focused on analyzing the features or characteristics of clones related to bugs [21]. A number of studies also analyzed the bug-proneness of clones from the perspective of inconsistencies due to late propagation [11], [22], [23].

Stability refers to the extent to which a code fragment remain unchanged. Less stable clones change more frequently and thus require comparatively more maintenance efforts than more stable clones. Stability of clones is one of the most widely investigated aspects of the impacts of clones on software systems [3], [24], [25], [26]. Although the existing studies have significant findings with respect to different aspects of the bug-proneness of clones, none of the existing studies investigated whether or to what extent the stability of clones is related to bug-proneness. Again, there are strong empirical evidences that inconsistent changes to clones and missing change propagation or late propagation are strongly related to the bug-proneness of clones [11], [22], [23], [27]. Also, different types of syntactic changes may have different impacts on change propagation and missing change propagation to clones are related to bugs and inconsistencies. Thus, it is important to investigate whether the types of syntactic changes to code clones contribute to their bug-proneness behaviour.

In particular, we represent the findings of our empirical study by answering the following research questions:

RQ1 *Is there any relationship between the stability and the bug-proneness of code clones? If yes, is the relationship significant?*

RQ2 *To what extent the association between the bug-proneness and stability of different types of clones are different?*

RQ3 *Do the frequencies of changes of different significance levels differently affect the bug-proneness of clones?*

In this study we investigate the relationships between the stability and the bug-proneness of clones using fine-grained change information from software repositories. The primary contributions of the paper are as follows:

(1) We investigate the relationships between the stability of clones and their bug-proneness. Although stability of clones is one of the most widely studied aspects of the impact of clones on software systems, none of the existing studies investigated whether or to what extent the stability of clones is related to the bug-proneness. Clone-induced inconsistencies and bugs are of important concern regarding the impacts of clones on software systems. Our investigation aims in revealing the relationships between stability and bug-proneness of clones. This is important whether stability is an influencing factor for clones to be bug-prone and vice-versa.
(2) Different types of clones may have a different degree of association with their corresponding stability. We carry out a clone type-centric analysis of the relationship between the stability and bug-proneness of clones.
(3) Although different types of syntactic changes have different impact on the software systems, none of the existing studies consider fine-grained change types while investigating the bug-proneness of clones. We consider fine-grained syntactic change types and their levels of change significance for in-depth analysis of the relationships between the stability and bug-proneness of clones.

We analyze the bug-proneness of clones from a new perspective. The study aims to relate the stability of clones as a potential influencing factor to the bug-proneness of clones. The findings of the study have the potentials to help in identifying and prioritizing clones for clone management activities such as clone refactoring and tracking.

The rest of the paper is organized as follows: Section II defines and explains key terminology in context of our study. Section III briefly describes the taxonomy of changes used in this study. Section IV represents the experimental settings and steps used in the study including the subject systems used, change extraction and classification procedure, and the metrics we measure. Section V represents the experimental results. Potential threats to the validity of the study is represented in section Section VI. Section VII discusses related works followed by the conclusion in Section VIII.

II. Terminology

Clones and Clone Types: Clones are exact or similar code fragments. Here, code *fragment* refers to a contiguous block of code. Code clones are of four types: Type 1, Type 2, Type 3 and Type 4 clones [28].

- Type 1 Clone: Type 1 clones are exact copies of code fragments except differences in layout, whitespace and comments.
- Type 2 Clone: In addition to Type 1 differences, Type 2 clones have differences in data types and identifier names.
- Type 3 Clone: Type 3 clones are created by addition, deletion and modification of statements or lines to Type 1 or Type 2 clones.
- Type 4 Clone: Type 4 clones refer to the semantically or functionally identical code fragments with different syntactic implementations.

Clones Pair and Clone Class: A pair of code fragments form a clone pair when they are clones of each other. A set of two or more code fragments in which any two code fragments belong to a clone pair are represented as clone class. In this study, we consider exact (Type 1) and near-miss (Type 2 and Type 3) clones for our analysis at method level granularity.

Change Significance: Fluri and Gall [29] in their taxonomy of fine-grained source code change, assign a level of *significance* to each of the change types. The *significance* level defines the extent of the impact of a change *i.e.,* the extent of changing system functionality and the likelihood of a change to affect other related entities for the required changed propagation. The higher the significance level, the higher impacts of the change will be on the related source code elements. They define four different levels *(low, medium, high, crucial)* of significance for fine-grained syntactic change types. Here, *crucial* is the highest level of change significance and *low* is the lowest level of change significance.

III. Taxonomy of Software Change at Method Level

Software systems evolve through different kinds of changes. For our study, we consider the taxonomy of change proposed by Fluri and Gall [29] which is a comprehensive taxonomy for fine-grained source code change and it defines the significance levels of changes. As our clone analysis is at method level granularity, we consider only the method level changes in the taxonomy proposed by Fluri and Gall. The taxonomy of changes used in this empirical study is presented in Table I.

Changes to individual methods are referred to as method-level changes. Changes to methods are further divided into two groups, changes to method declaration and changes to method body, based on where the changes occur in the methods. Changes to method declaration part (*method signature*) include changes in accessibility, overridability, method renaming, parameter change and changes to return type. Changes to parameters include addition, deletion, renaming, reordering and parameter type change. Changes to *method body* comprise changes to statements and structure statements (*e.g.,* loop, branching). Statements might be added, deleted, modified and reordered. Each of these fine-grained change types is assigned a level of significance based on their likelihood of affecting other code entities and the extent they modify the functionality of the system as proposed by Fluri and Gall.

IV. Experimental Setup

This section describes the experimental setup of our empirical study including the subject systems used, methodologies for identifying buggy commits, change extraction and classification,

TABLE I: TAXONOMY OF METHOD-LEVEL CHANGES WITH SIGNIFICANCE (adapted from [29])

Part	Change Group	Change Type	Significance
Method Declaration	Accessibility	Accessibility Increase	Medium
		Accessibility Decrease	High
	Overridability	Add Method Overridability *final*	Crucial
		Delete Method Overridability	Low
	Parameter	Parameter Insert	Crucial
		Parameter Delete	Crucial
		Parameter Ordering	Crucial
		Parameter Renaming	Medium
		Parameter Type Change	Crucial
	Method Name	Method Renaming	High
	Return Type	Return Type Insert	Crucial
		Return Type Delete	Crucial
		Return Data Type Chang	Crucial
Method Body	Statement	Statement Insert	Medium
		Statement Delete	Medium
		Statement Update	Low
		Statement Re-ordering	Low
	Structure Statement	Condition Expression Change	Medium
		Statement Parent Change	Medium
		Alternative- part (*else*) Insert	Medium
		Alternative- part (*else*) Delete	Medium

clone detection and the measurement of comparative stabilities of buggy and non-buggy clones. The following subsections describe the details of the experimental settings of our study.

A. Subject Systems

This study is based on five open source systems implemented in Java with diversified size and application domain. Table II briefly represents the features of the software systems including the application domain, size in lines of code (LOC) and the total number of revisions. The size of the systems represents the lines of code (LOC) in the last revision of the systems counted after removal of comments and pretty-printing.

TABLE II: SUBJECT SYSTEMS

Systems	Type	Size (LOC)	#Revision
DNSJava	DNS Protocol	20831	1679
JabRef	Bibliography Manager	153952	3718
Carol	Driver Application	13213	2237
Ant-Contrib	Web Server	79434	177
OpenYMSG	Open Messenger	8821	233

B. Detecting Bug-fix Commits

To identify bug-fix commits of a candidate systems, we extract the SVN commit messages by applying *SVN log* command. A commit message describes the objective of the associated commit and thus can be used to infer whether the commit is a bug-fix commit. We apply the heuristics proposed by Mockus and Votta [30] on the commit messages to automatically identify bug-fix commits. This approach for identifying bug-fix commits have been used in different earlier studies [11], [18], [31], [32]. This technique identifies bug-fix commits based on the occurrence of keywords in the commit message from a predefined set. For example, if a commit message contains the word "bug", it will be classified as a bug-fix commit. This heuristic based approach may sometimes results in false positives due to incorrect classification of commits as bug-fix commits. However, earlier study [11] shows that this approach can identify bug-fix commits with 87% accuracy. Once the bug-fix commits are listed, we identify the bug-fix commits where the cloned fragments have changed. When any clone fragment is changed in a bug-fix commit, it is reasonable to infer that changes to that clone is necessary to fix the bug. We analyze and identify all the cloned methods that are related to bug-fixes. We then analyze the stability considering fine-grained change types associated with each of the bug-related clone fragments to measure the extent of the relationship between stability and bug-proneness.

C. Experimental Steps

We carry out our experimental analysis through some processing steps. Figure 1 shows the schematic diagram of the overall experimental analysis process. We briefly describe the experimental steps as follows:

1) Preprocessing: For our study, we first extract all the revisions of the subject systems from their corresponding *SVN* repositories [33]. As our analysis focus on the changes to source code only, we remove comments from the source files. Pretty-printing of the the source files are then carried out to eliminate the formatting differences using the tool *ArtisticStyle* [34]. We extract the file modification history using *SVN diff* command to list added, modified and deleted files in successive revisions. This information is used to exclude the unchanged files during change analysis to speed up the process.

2) Method Extraction and Origin Analysis: To analyze the changes to cloned methods throughout all the revisions, we extract method information from the successive revisions of the source code. We store the method information (file path, package name, class name, signature, start line, end line) in a database to use for mapping changes to the corresponding methods. Again, SVN keeps track of the files that are added, deleted or modified and the history of changes to individual file content are preserved as modification of lines. This line-level change information is not sufficient to describe the evolution of source code entities at higher granularity levels such as classes or methods. As a result, to map changes to methods throughout the development cycle, we need to map the methods across the revisions. Therefore, we carry out origin analysis of the methods on the revisions of the systems. We used same approach for origin analysis as presented in study by Lazano and Wermelinger [9]. Here, if a method is relocated in the same file or if the method signature is changed, the method is mapped by string comparison with the candidate methods in new file using *Strike A Match* algorithm [35]. The origin mapping information is used to map the classified changes and cloning information back to the corresponding methods.

3) Change Extraction and Classification: The change analysis system in this study is implemented in Java based on the change extraction and classification core of *ChangeDistiller* [29]. The system imports copies of changed files from successive versions and uses JDT API of Eclipse for the extraction of methods and extracting the differences between the copies

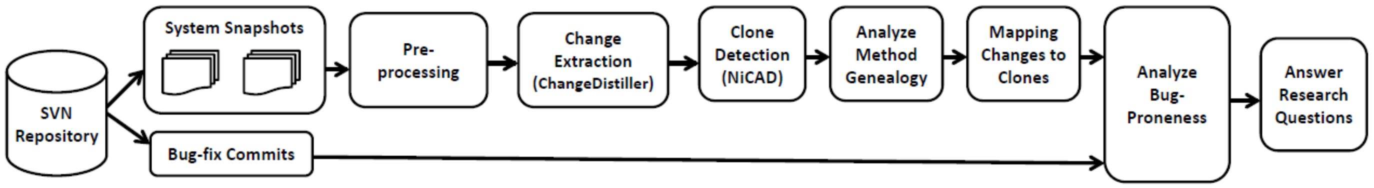

Fig. 1: Overall Analysis Process

of each of the files in any two successive revisions. The details of change extraction and classification are as follows:

- *Change Extraction:* Code changes are extracted using *ChangeDistiller* classifier. *ChangeDistiller* extracts changes by taking differences between two versions of ASTs of the same file. The differences are represented as a sequence of tree-edit operations. The generic operations contain insert, delete, move and update operations on the nodes in the AST. The tree-edit operations encoded as edit scripts are then processed by *ChangeDistiller* to classify extracted changes to fine-grained change types. We have customized the *ChangeDistiller* classifier to suit for analyzing local repository exported from SVN. To extract source code changes, two successive versions of the same file are selected from the source repository and then are passed to the differencing engine of the change classifier. The extracted changes are then passed to the classifier for classification. The process is repeated for all changed files (identified by SVN *diff*) and for all the revisions of the subject systems.
- *Change Classification:* Changes extracted by AST-differencing of two successive revisions of source code files are classified into fined-grained changes to source code entities. Changes are classified according to the defined taxonomy and are assigned the corresponding levels of significance. For the analysis, classified changes are mapped to the corresponding source code entities based on the information extracted during the origin analysis.

4) Mapping Change Data: After classification, the classified changes are mapped to their corresponding source code entities (methods) with the help of extracted origin mapping information. We preserve the extracted, classified and mapped changes into a database to measure metrics for the bug-proneness of clones at the method level granularity.

5) Clone Detection: There are a great many clone detection tools and techniques [36], [37], [38], [39], [40], [41], [42] available including a recent one using deep learning [43]. However, for this study we use the hybrid clone detection tool NiCad [44], [45]. NiCad is reported to have higher level of precision and recall [46] and supports the detection of both exact (Type 1) and near-miss (Type 2 and Type 3) clones. We run NiCad on all revisions of the subject systems to detect clones at method level granularity. We then store the clone information in the database. Table III lists the parameter settings for NiCad used for this study.

TABLE III: NiCad SETTINGS FOR THE STUDY

Parameters	Values
Minimum Size	5 lines
Maximum Size	500 lines
Granularity	Method
Threshold	0% (Type-1, Type-2), 30% (Type-3)
Identifier Renaming	blindrename (Type-2, Type-3)

6) Mapping Changes to Clones: We use the extracted method genealogies to map method information to the detected clones in each revision of the software systems. We also map classified fine-grained changes to the clones in each revision. Using the list of identified bug-fix commits we separate the list of buggy and non-buggy clones. Methods that were changed in any of the bug-fix commits are considered as buggy clones while other cloned methods are considered as non-buggy clones.

D. Measuring the Change Frequencies

Once the mapped change information for buggy and non-buggy clones are available, we measure the stability (frequency of changes) for buggy and non-buggy clones. We measure stability with respect to different clone types considering the different levels of significance of fine-grained syntactic changes. We measure the frequency of changes as follows:

Let R_b be the set of bug-fix commits and $S = \{low, medium, high, crucial\}$ be the set of different levels of change significance. Let M_b and M_n are the sets of buggy and non-buggy cloned methods respectively. Then, we measure the stability (frequency of changes) of buggy and non-buggy clones using the following two equations:

(i) The frequency of changes to buggy cloned methods (CF_b) is measured as

$$CF_b = \frac{\sum_{m_i \epsilon M_b, s \epsilon S} CC_s(m_i)}{\sum_{m_i \epsilon M_b} AVGLOC(m_i)} \quad (1)$$

(ii) The frequency of changes to non-buggy cloned methods (CF_n) is measured as

$$CF_n = \frac{\sum_{m_i \epsilon M_n, s \epsilon S} CC_s(m_i)}{\sum_{m_i \epsilon M_n} AVGLOC(m_i)} \quad (2)$$

In Equation 1 and Equation 2,

- $CC_s(m_i)$ represents the total number of changes to a cloned method m_i with change significance level s where $s \epsilon S$.
- $AVGLOC(m_i)$=$\frac{\sum_{r \epsilon R_{m_i}} LOC_r(m_i)}{|R_{m_i}|}$ represents the average size (LOC) of a cloned method. Here, $LOC_r(m_i)$ refers to the size (LOC) of the cloned method in revision

TABLE IV: COMPARATIVE FREQUENCIES OF CHANGES OF DIFFERENT LEVELS OF SIGNIFICANCE FOR BUGGY AND NON-BUGGY CLONES OF ALL TYPES.

Change Significance→	Low		Medium		High		Crucial		Overall	
Systems ↓	CF_b	CF_n	CF_b	CF_n	CF_b	CF_n	CF_b	CF_n	CF_b	CF_n
DNSJava	1.6546	0.5145	1.4196	0.5177	0.1471	0.1150	0.0723	0.0254	3.2936	1.1726
JabRef	0.7213	0.3496	0.4198	0.1768	0.0033	0.0031	0.0267	0.0163	1.1710	0.5458
Carol	0.7505	0.2740	0.6515	0.2542	0.0178	0.0153	0.0323	0.0248	1.4521	0.5684
Ant-Contrib	0.4325	0.2010	0.3249	0.1484	0.0172	0.0031	0.0038	0.0371	0.7785	0.3896
OpenYmsg	0.4211	0.2280	0.4044	0.0909	0.0069	0.0000	0.0014	0.0014	0.8338	0.3803
***p*-value**	**0.0366**		0.0601		0.2113		0.7565		0.0601	
Cohen's *d*	1.3129		1.1964		0.2016		0.2828		1.1651	
PS	0.82		0.80		0.55		0.58		0.80	

CF_b=Frequency of Changes in Buggy Cloned Methods
CF_n=Frequency of Changes in Non-Buggy Cloned Methods, PS=Probability of Superiority

$r \epsilon R_{m_i}$ where R_{m_i} is the set of revisions a method m_i was cloned.

- A cloned method is considered buggy (*i.e.,* $m_i \epsilon M_b$) if $R_{m_i} \cap R_b \neq \phi$, *i.e.,* the method m_i has changed in at least one of the bug-fix commits. Otherwise, the method is a non-buggy (*i.e.,* $m_i \epsilon M_n$) cloned method.

Here, the change frequencies of buggy and non-buggy clones simply represent the average number of changes per line of buggy and non-buggy cloned methods respectively. The lower the frequency of changes of a clone fragment, the higher will be the stability of that clone fragment.

V. Results

We present our experimental results in the following subsections to answer the research questions we defined in Section I.

Answer to RQ1*: Is there any relationship between the stability and the bug-proneness of code clones? If yes, is the relationship significant?*

Importance: Stability of clones is one of the most widely studied aspects of analyzing the impacts of clones on software systems. Studies [9], [25], [26], [16] show that clones are often comparatively less stable meaning that clones change more frequently than non-cloned code. It is intuitive that the more frequently a code fragment is changed, the more maintenance efforts it will require. Again, clone fragments changing more frequently are likely to increase the chance of missing change propagations and thus may introduce inconsistencies or bugs in the software systems. Consequently, it is important to investigate if and to what extent the stability of clones are related to their bug-proneness.

Methodology: To analyze the relationships between the stability and the bug-proneness of clones, we measure the frequencies of changes for buggy and non-buggy clones as described in Section IV-D for each of the subject systems. First we determine the list of bug-fix commits by analyzing the commit messages from SVN repositories (Section IV-B). Then we identify the changed cloned methods from a selected clone type. Then we count the number of changes of different significance levels for each of the methods selected. We also count the average size (LOC) of each of the cloned methods in the list. We measure the total number of changes and the total code size (LOC) for both buggy and non-buggy cloned methods. Then we measure the corresponding frequencies of changes as the ratio of total number changes to total code size in LOC for both buggy and non-buggy clones for comparison. We measure the frequency of changes regarding each level of significance of change. We consider all clones without differentiating their types for answering RQ1.

Table IV shows the frequencies of changes for buggy and non-buggy clones for different levels of change significance. Here, we observe that change frequencies for buggy clones for all levels of change significance and for all the systems are higher (*i.e.,* CF_b>CF_n) than the corresponding frequencies of changes of non-buggy clones (except for the *crucial* significance in OpenYmsg). This shows that buggy clones tend to be less stable than non-buggy clones. To analyze whether there is any statistically significant relationship between the stability and bug-proneness of clones, we carry out Mann-Whitney Wilcoxon (MWW) test (two-tailed, at <0.05) [47]. From Table IV, we see that the p-value for *low* level of change significance is 0.0366 which is less than 0.05. This implies that the frequency of changes for buggy clones is significantly higher than the frequency of changes in non-buggy clones regarding changes of *low* significance. However, for other levels of change significance (*medium, high and crucial*), although the value of frequencies of changes for buggy clones are higher than that of non-buggy clones, the p-values are greater than 0.05. This indicates that change frequencies for buggy and non-buggy clones are not significantly different for those cases.

Again, the Mann-Whitney Wilcoxon test examines whether the findings are likely due to chance and may not alone fully express the magnitude of differences found. Thus, we also calculate the *effect size* to analyze the magnitude of differences in stability of buggy and non-buggy clones. Effect size calculates the standardized mean difference between two data sets. We measure the effect size (Cohen's d [48]) from the comparative frequencies of changes to buggy and non-buggy clones as shown in Table IV. We see that the values of the effect size for both *low* and *medium* significance levels are 1.3129 and 1.1964 respectively which belong to 'large' category as they are above 0.8. The effect size for *high* and *crucial* significance levels are 'small' with values 0.2016 and 0.2828 respectively. For easier interpretation of the effect size, we also convert the effect size values to the probability of superiority or 'Common Language Effect Size' [49] as shown in Table IV. The probability of superiority value for *low* significance level is 0.82 meaning that if selected randomly, there is 82% chance that buggy clones will have higher change frequency than non-buggy clones. This probability values are lower for changes of *high* (0.55) and *crucial* (0.58) significance and closer to 0.5 (0.5 refers to equal likelihood of having higher change frequencies for buggy and non-buggy clones). We also analyzed the comparative stability of buggy and non-buggy clones by aggregating changes of all levels of significance (shown in the 'overall' column). Here, we observe that CF_b>CF_n for all the subject systems and the effect size is large (1.1651). The probability of superiority is 0.8 *i.e.,* in 80% cases buggy clones likely to have higher frequency of changes than non-buggy clones. Thus, buggy clones tend to be less stable than the non-buggy clones.

Summary- *According to our experimental results, buggy clones tend to have higher frequency of changes compared to non-buggy clones* i.e., *buggy clones are less stable than non-buggy clones. Thus, the stability and the bug-proneness of clones are related and this relationship is statistically significant for changes of low significance level.*

Answer to RQ2: *To what extent the bug-proneness of different types of clones are associated with their stability?*

Importance: Different types of clones exhibit different degree of bug-proneness [18]. Similarly, stability of different types of clones may vary with the changes of different levels of significance. Thus, an in-depth analysis of the relationships between the stability of clones and their bug-proneness should also investigate the degree of the relationships regarding different clones types. This clone type centric analysis is likely to reveal how the relationship between stability and bug-proneness of clones varies for different types of clones. This analysis is important to identify what types of clones to be comparatively more bug-prone and need to be prioritized in clone management activities.

Methodology: To answer RQ2, we analyze the comparative frequencies of changes of different levels of significance for buggy and non-buggy clones considering clone types separately. We measure the frequency of changes for buggy and non-buggy clones in the same way as in RQ1. However, we measure the stability metrics for distinct types of clones (Type 1, Type 2 and Type 3) separately. The results for the comparative frequencies of changes of buggy and and non-buggy clone for Type 1, Type 2 and Type 3 clone are presented in Table V, Table VI, Table VII respectively.

For Type 1 clones (Table V), the frequency of changes for buggy clones are higher than that of non-buggy clones for all subject systems when we consider changes of *low* and *medium* significance (*i.e.,* CF_b>CF_n). However, for changes of *high* significance in JabRef and Carol and for changes of *crucial* significance in Ant-Contrib and OpenYmsg we observe CF_b<CF_n. We observe that for changes of *low* and *medium* significance, the likelihood of buggy Type 1 clones to have higher frequencies than non-buggy Type 1 clones are 76% and 78% respectively. However, we do not observe any statistically significant relationship between the stability and the bug-proneness for Type 1 clones.

For Type 2 clones (Table VI), we observe that the frequency of changes for buggy clones are higher than that of non-buggy clones for all the subject systems regarding changes of *low* and *medium* significance. For comparative frequencies of buggy and non-buggy Type 2 clones with changes of *low* and *medium* significance, we get the p-values 0.0366 and 0.0121 respectively from Mann-Whitney Wilcoxon (MWW) test (two-tailed, at <0.05). These p-values imply that the frequency of changes in buggy Type 2 clones are significantly higher than the frequency of changes in non-buggy Type 2 clones regarding changes of *low* and *medium* significance. In addition, for changes of *low* and *medium* significance, the values of effect size are 'large' (1.6283 and 1.6916 respectively) with the corresponding likelihood of 87% and 88% that buggy Type 2 clones have higher frequency of changes than non-buggy Type 2 clones. However, we did not observe any statistically significant differences between CF_b and CF_n for changes of *high* and *crucial* significance.

As shown in Table VII, the frequencies of changes for Type 3 buggy clones are higher than that of non-buggy Type 3 clones for all the subject systems regarding changes of *low* and *medium* significance. For Type 3 clones with changes of *low* significance, the p-value for Mann-Whitney Wilcoxon (MWW) test (two-tailed, at <0.05) is 0.0366. This implies that for changes of *low* significance, the frequency of changes in buggy Type 3 clones are significantly higher than that of non-buggy Type 3 clones. The effect size d for comparative frequencies for changes for buggy and non-buggy Type 3 clones is also 'large' (1.2916) for changes of *low* significance. For *low* significance, buggy Type 3 clones have 82% likelihood of having higher frequency of changes than non-buggy Type 3 clones.

Figure 2 represents the comparative likelihood (%) of buggy clones to have higher frequencies for changes than non-buggy clones with respect to different clones types. We use the probability of superiority from the effect size analysis for comparative change frequencies of buggy and non-buggy clones. We consider probabilities of superiority for all four levels of change significance with respect to each type of clones. From Figure 2 we see that for Type 1 clones, the probability of superiority values regarding changes of *low* and *medium* significance are 76% and 78% respectively. On the other hand Type 2 clones have the highest likelihood (87% and 88% respectively) of buggy clones to have higher frequency of changes than non-buggy clones regarding changes of *low* and *medium* significance. Type 3 clones also have 82% and 88% likelihood of buggy clones to have higher frequency of changes than non-buggy clones. However, for all clone types while considering the changes of *high* and *crucial* significance, the likelihood of buggy and non-buggy clones are closer to 50%. This indicates that the probability of buggy and non-buggy clones to have higher frequency of changes are similar.

TABLE V: COMPARATIVE FREQUENCIES OF CHANGES OF DIFFERENT LEVELS OF SIGNIFICANCE FOR BUGGY AND NON-BUGGY CLONES OF TYPE 1.

Change Significance→	Low		Medium		High		Crucial		Overall	
Systems ↓	CF_b	CF_n	CF_b	CF_n	CF_b	CF_n	CF_b	CF_n	CF_b	CF_n
DNSJava	1.6438	0.4373	1.4155	0.4373	0.0918	0.0435	0.0787	0.0239	3.2297	0.9422
JabRef	0.6710	0.3409	0.3861	0.1571	0.0021	0.0026	0.0190	0.0082	1.0783	0.5088
Carol	0.3235	0.2102	0.5202	0.1564	0.0087	0.0587	0.0393	0.0391	0.8918	0.4643
Ant-Contrib	0.4299	0.2571	0.3277	0.1959	0.0162	0.0041	0.0011	0.0490	0.7749	0.5061
OpenYmsg	0.4422	0.2262	0.2802	0.0524	0.0103	0.0000	0.0000	0.0024	0.7326	0.2810
***p*-value**	0.0601		0.0601		0.6745		0.8337		0.0601	
Cohen's *d*	1.0492		1.1062		0.1226		0.1147		1.0379	
PS	0.76		0.78		0.52		0.52		0.76	

CF_b=Frequency of Changes in Buggy Cloned Methods
CF_n=Frequency of Changes in Non-Buggy Cloned Methods, PS=Probability of Superiority

TABLE VI: COMPARATIVE FREQUENCIES OF CHANGES OF DIFFERENT LEVELS OF SIGNIFICANCE FOR BUGGY AND NON-BUGGY CLONES OF TYPE 2.

Change Significance→	Low		Medium		High		Crucial		Overall	
Systems ↓	CF_b	CF_n	CF_b	CF_n	CF_b	CF_n	CF_b	CF_n	CF_b	CF_n
DNSJava	1.0859	0.3979	0.9594	0.3650	0.1186	0.1885	0.0422	0.0299	2.2061	0.9814
JabRef	0.8192	0.5251	0.4398	0.2590	0.0000	0.0035	0.0751	0.0987	1.3341	0.8863
Carol	1.3994	0.2267	1.0694	0.2713	0.0293	0.0167	0.0146	0.0056	2.5127	0.5204
Ant-Contrib	0.4200	0.0455	0.3800	0.1667	0.0000	0.0000	0.0000	0.0606	0.8000	0.2727
OpenYmsg	0.4211	0.2880	0.4044	0.0909	0.0069	0.0000	0.0014	0.0014	0.8338	0.3803
***p*-value**	**0.0366**		**0.0121**		1.000		0.6031		0.0949	
Cohen's *d*	1.6283		1.6916		-0.1567		-0.3414		1.5519	
PS	0.87		0.88		0.55		0.58		0.87	

CF_b=Frequency of Changes in Buggy Cloned Methods
CF_n=Frequency of Changes in Non-Buggy Cloned Methods, PS=Probability of Superiority

TABLE VII: COMPARATIVE FREQUENCIES OF CHANGES OF DIFFERENT LEVELS OF SIGNIFICANCE FOR BUGGY AND NON-BUGGY CLONES OF TYPE 3.

Change Significance→	Low		Medium		High		Crucial		Overall	
Systems ↓	CF_b	CF_n	CF_b	CF_n	CF_b	CF_n	CF_b	CF_n	CF_b	CF_n
DNSJava	1.2390	0.7305	0.9372	0.8774	0.3268	0.3449	0.0590	0.0292	2.5620	1.9820
JabRef	0.9494	0.3227	0.5978	0.1779	0.0039	0.0027	0.0357	0.0277	1.5869	0.5310
Carol	0.7698	0.2874	0.6624	0.2558	0.0186	0.0135	0.0312	0.0233	1.4820	0.5800
Ant-Contrib	0.3735	0.1159	0.3550	0.0454	0.0078	0.0000	0.0029	0.0202	0.7391	0.1814
OpenYmsg	0.3918	0.2783	0.5623	0.0569	0.0000	0.0000	0.0000	0.0000	0.9541	0.3353
***p*-value**	**0.0366**		0.0949		0.6744		0.4654		0.0949	
Cohen's *d*	1.2916		1.1945		-0.0054		0.2903		1.0386	
PS	0.82		0.80		0.50		0.55		0.76	

CF_b=Frequency of Changes in Buggy Cloned Methods
CF_n=Frequency of Changes in Non-Buggy Cloned Methods, PS=Probability of Superiority

From the above clone type based analysis, we observe that although the frequency of changes tend to be higher in buggy clones than in the non-buggy clones for both exact (Type 1) and near-miss(Type 2 and Type 3) clones, the bug-proneness of Type 2 and Type 3 clones have stronger relationships with their stability. The significance of the relationship between the bug-proneness of Type 2 and Type 3 clones are mostly influenced by changes of *low* and *medium* significance.

Summary- *Our results on clone type centric analysis show that the bug-proneness of Type 2 and Type 3 clones are significantly related to the stability of clones and this relationship is mostly influenced by the changes of low and medium significance. The bug-proneness of Type 1 clones does not exhibit significant relationship with the stability.*

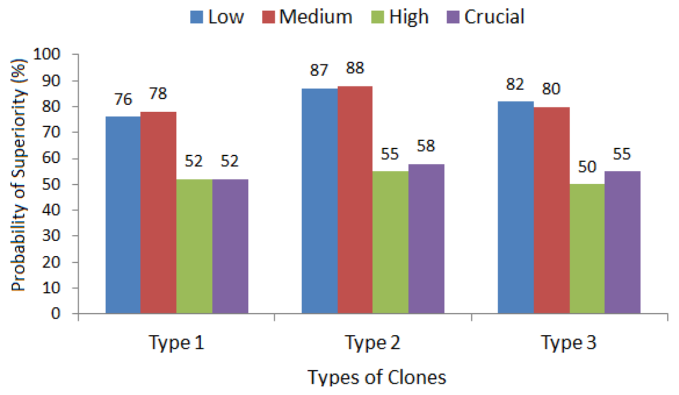

Fig. 2: Likelihood (%) buggy clones having higher frequencies of changes than non-buggy clones regarding different types of clones.

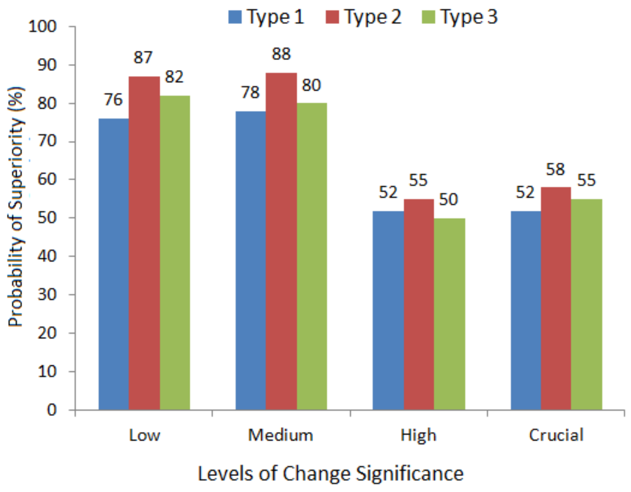

Fig. 3: Likelihood (%) buggy clones having higher frequencies of changes than non-buggy clones regarding different levels of change significance.

Answer to RQ3: *Do the frequencies of changes of different significance levels differently affect the bug-proneness of clones?*

Importance: Changes of different levels of significance have different impacts on software systems both in terms of the modification of the system functionality and the extents of required change propagation. Thus, for better understanding the relationships between bug-proneness and stability of clones, we need to investigate how changes of different levels of significance influence this relationship. Therefore, we analyze the comparative stability of buggy and non-buggy clones with respect to each of the four levels of change significance.

Methodology: To answer RQ3, we consider the comparative frequencies of changes for the buggy and non-buggy clones form Table V, Table VI and Table VII regarding each type of change significance. For changes of *low* significance and for all types (Type 1 , type 2 and Type 3) of clones, buggy clones have higher frequencies of changes than non-buggy clones. However, for changes of *low* significance, the comparative frequencies of changes for buggy and non-buggy clones are statistically significant for only for Type 2 and Type 3 clones. For changes of *medium* significance, all types of clones tend to have higher frequencies of changes for buggy clones compared to non-buggy clones. However, only Type 2 clones show a statistically significant difference in the frequencies of changes in buggy and non-buggy clones. For changes of *high* and *crucial* significance, none of the clone types exhibit any significant differences between the frequency of changes in buggy and non-buggy clones. Thus, changes of *low* and *medium* significance likely to influence the relationship between the stability and the bug-proneness of clones.

Figure 3 represents the comparative likelihood (%) of buggy clones to have higher frequencies for changes than non-buggy clones with respect to different levels of changes significance. We use the probability of superiority from the effect size analysis for comparative change frequencies of buggy and non-buggy clones. From Figure 3 we see that for changes of *low* significance, the probability of superiority values regarding Type 1, Type 2 and Type 3 clones are 76%, 87% and 82% respectively. On the other hand, for changes of *medium* significance, the probability of superiority values regarding Type 1, Type 2 and Type 3 clones are 78%, 88% and 80% respectively. These imply that the frequencies of changes to buggy clones are comparatively higher than that of non-buggy clones regarding changes of *low* and *medium* significance for all clone types. However, Type 2 clones have the highest probability values for buggy clones to have higher frequency of changes than non-buggy clones for changes of *low* and *medium* significance. For changes of *high* and *crucial* significance, none of the clone types exhibit higher likelihood of buggy clones to have higher frequencies of changes than non-buggy clones. Here the negative value for d indicates the direction of mean difference.

Type 2 clones have the highest likelihood (87% and 88% respectively) of buggy clones to have higher frequency of changes than non-buggy clones regarding changes of *low* and *medium* significance. Type 3 clones also have 82% and 88% likelihood of buggy clones to have higher frequency of changes than non-buggy clones. However, for all clone types while considering the changes of *high* and *crucial* significance, the likelihood of buggy and non-buggy are closer to 50% which indicates that the probability of buggy and non-buggy clones to have higher frequency of changes are similar.

Summary- *The bug-proneness of clones tend exhibit stronger relationship with the stability regarding the changes of low and medium significance. Thus, the relationship between the stability and the bug-proneness of clones is most likely to be influenced by changes of low and medium significance.*

VI. Threats to Validity

The change analysis in this study is based on the change extraction and classification engine of the *ChangeDistiller*. Although *ChangeDistiller* is reported to have good performance, the validity of the outcomes of the study is dependent upon the accuracy of the core classifier used.

The detection of bug-fix commits in our study is based on the technique proposed by Mockus and Votta [30]. Same technique has also been used in several earlier studies [11], [18], [31], [32], [50]. Although this heuristics based technique may result in some incorrect identification of bug-fix commits, earlier study by Barbour *et al.* [11] shows that the probability is low. Their study reports that the accuracy of the technique to identify bug-fix commits is 87%.

We used NiCad, a hybrid clone detection tool which detects Type 1, Type 2 and Type 3 clones with high precision and recall. Again, different settings for clone detection tool may result in different stability scenarios because of the variations in clone detection results. This is termed as *confounding configuration choice problem* [51]. However, the NiCad settings we used are considered standard [1] and are close enough to the optimal configuration settings identified in recent study [51] for NiCad to detect clones in Java systems. This is likely to mitigate the potential adverse effects of the configuration settings on our findings.

We selected subject systems with diversified size, number of revisions, length of evolution and application domain to avoid potential biasing. However, due to the limitation of existing change classifier our study is limited to Java systems only. Although Java is considered to be a widely used language with a comprehensive set of language features, inclusion of subject systems of other languages might help in more generalization of the findings.

VII. Related Work

Many studies have investigated the bug-proneness of cloned code from different perspectives. Wagner *et al.* [20] investigated the relationship between Type 3 clones and faults. This study shows that a considerable proportion (17%) of Type 3 clones are associated with faults. Mondal *et al.* [18] investigated the bug-proneness of different types of clones and reported that Type 3 clones are more bug-prone compared to Type 1 and Type 2 clones. Although, this study gives important insights regarding the bug-proneness of clones of different types, the study does not consider fine-grained change information for the analysis as we did.

Mondal *et al.* in a recent study [31] observed that change-prone clones are not necessarily to be bug-prone. They measured change frequency as the number of commits a particular clone fragment was changed before a given bug-fix commit. Our study on the other hand considers fine-grained change types and their significance. Our change frequency metrics consider the actual number of fine-grained changes per line of cloned code. Thus, our analysis of relationships between stability and bug-proneness is different from the study by Mondal *et al.* [31].

Li *et al.* [13] proposed the tool CP-Miner that uses data mining techniques to detect copy-baste clones and bugs in large scale software systems. Li and Ernst [14] studied bug-proneness of clones and developed a tool called CBCD based on their study. For a given buggy code fragment, CBCD searches semantically identical copies of the given code fragment. Inoue *et al.* [52] developed a tool called CloneInspector to detect inconsistent changes to code clones and associated latent bugs in software systems.

Barbour *et al.* [11] examined late propagation of clones (Type 1 and Type 2) and investigated the extent to which different types of late propagation are related to bugs and inconsistencies in software systems. Aversano *et al.* [22] reported that clones evolve consistently in most cases while late propagation may introduce faults. Thummalapenta *et al.* [23] show that clones with late propagation are more bug-prone than others.

Xie *et al.* [19] studied the fault-proneness of clones with respect to mutation and migration of clones during evolution. They conclude that both mutation of a clone group to Type 2 or Type 3 clones and increasing distance between clone fragments in a clone group increase the risk for faults. Steidl and Göde [21] proposed a machine learning approach to investigate how different features of clones can be used to automatically identify incomplete bug-fixes. Göde and Koschke [53] reported that changes to clones are mostly infrequent and a small proportion (14.8%) of changes are unintentionally inconsistent.

Sajnani *et al.* [54] analyzed the comparative bug-proneness of cloned and on-cloned code on 31 Java projects. They report that the density of bugs are comparatively less in clones than non-cloned code. Rahman *et al.* [6] observed that clones may be less defect-prone than non-cloned code which is opposite to the findings of the study by Juergens *et al.* [8].

The existing studies reveal important insights regarding the techniques for the detection [13], [17] of bug-prone clones, identifying features and types of bugs [17] and analyzing bug-proneness of different types of clones [18] from the perspective of consistent evolution. However, none of the existing studies consider fine-grained syntactic changes despite the types of changes mostly define the extent of change propagation to related cloned and non-cloned fragments. In this study, we extract fine-grained change information and analyze the relationship between the stability and the bug-proneness of clones. We evaluate stability as a potential influencing factor to the bug-proneness of clones and analyze the bug-proneness of Type 1, Type 2 and Type 3 clones.

VIII. Conclusion

This study investigates the relationships between the stability and the bug-proneness of clones considering fine-grained syntactic change types and their significance extracted using ChangeDistiller. We automatically identify the bug-fix commits by analyzing the commit messages from the SVN repositories. We detect Type 1, Type 2 and Type 3 clones using the hybrid cloned detection tool NiCad. Clones that change in any of the bug-fix commits are identified as buggy clones. Our experimental results on five diverse open source Java systems show that buggy clones tend to have higher frequency of changes compared to non-buggy clones. This relationship is often significant especially when changes of *low* and *medium* significance are considered. Again, the stabilities of Type 2 and Type 3 clones have comparatively stronger association with the bug-proneness than Type 1 clones. The relationships between the stability and bug-proneness of different types of clones are more likely influenced by the changes of *low* and *medium* significance. Our findings regarding the relationships between the stability and bug-proneness of clones has the potential to identify bug-prone clones based on their change-proneness. Thus, the correspondence between the stability and bug-proneness of clones might be useful to identify and to prioritize important clones for management. Our experimental data are available on-line [55].

REFERENCES

[1] C. K. Roy and J. R. Cordy, "Near-miss function clones in open source software: An empirical study," *Soft. Maint. and Evol: Res. Pract.*, vol. 22, no. 3, pp. 165–189, 2010.

[2] K. Hotta, Y. Sano, Y. Higo, and S. Kusumoto, "Is duplicate code more frequently modified than non-duplicate code in software evolution?: An empirical study on open source software," in *Proc. IWPSE*, 2010, pp. 73–82.

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

[4] ——, "Is cloned code older than non-cloned code?" in *Proc. IWSC*, 2011, pp. 28–33.

[5] ——, "A study of consistent and inconsistent changes to code clones," in *Proc. WCRE*, 2007, pp. 170–178.

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

[7] C. Kapser and M. W. Godfrey, ""Cloning considered harmful" considered harmful: patterns of cloning in software," *Emp. Soft. Engg.*, vol. 13, no. 6, pp. 645–692, 2008.

[8] E. Juergens, F. Deissenboeck, B. Hummel, and S. Wagner, "Do code clones matter?" in *Proc. ICSE*, 2009, pp. 485–495.

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

[10] ——, "Tracking clones' imprint," in *Proc. IWSC*, 2010, pp. 65–72.

[11] L. Barbour, F. Khomh, and Y. Zou, "An empirical study of faults in late propagation clone genealogies," *Soft. Evol. and Proc.*, vol. 25, pp. 1139–1165, 2013.

[12] D. Chatterji, J. C. Carver, B. Massengil, J. Oslin, and N. A. Kraft, "Measuring the eficacy of code clone information in a bug localization task: An empirical study," *ESEM*, pp. 20–29, 2011.

[13] Z. Li, S. Lu, S. Myagmar, and Y. Zhou, "Cp-miner: Finding copy-paste and related bugs in large-scale software code," *IEEE TSE*, vol. 32, pp. 176–192, 2006.

[14] J. Li and M. D. Ernst, "CBCD: Cloned buggy code detector," in *Proc. ICSE*, 2012, pp. 310–320.

[15] M. S. Rahman and C. K. Roy, "A change-type based empirical study on the stability of cloned code," in *Proc. SCAM*, 2014, pp. 31–40.

[16] M. Mondal, M. S. Rahman, C. K. Roy, and K. A. Schneider, "Is cloned code really stable?" *Emp. Soft. Engg.*, Jul 2017.

[17] L. Jiang, Z. Su, and E. Chiu, "Context-based detection of clone-related bugs," in *Proc. FSE*, 2007, pp. 55–64.

[18] M. Mondal, C. K. Roy, and K. A. Schneider, "A comparative study on the bug-proneness of different types of code clones," in *Proc. ICSME*, 2015, pp. 91–100.

[19] S. Xie, F. Khomh, and Y. Zou, "An empirical study of the fault-proneness of clone mutation and clone migration," in *Proc. MSR*, 2013, pp. 149–158.

[20] S. Wagner, A. Abdulkhaleq, K. Kaya, and A. Paar, "On the relationship of inconsistent software clones and faults: An empirical study," in *Proc. SANER*, vol. 1, 2016, pp. 79–89.

[21] D. Steidl and N. Göde, "Feature-based detection of bugs in clones," in *Proc. IWSC*, 2013, pp. 76–82.

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

[23] S. Thummalapenta, L. Cerulo, L. Aversano, and M. D. Penta, "An empirical study on the maintenance of source code clones," *Emp. Soft. Engg.*, vol. 15, no. 1, pp. 1–34, 2010.

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

[25] ——, "Tracking clones' imprint," in *Proc. IWSC*, ser. IWSC '10, 2010, pp. 65–72.

[26] M. Mondal, C. K. Roy, M. S. Rahman, R. K. Saha, J. Krinke, and K. A. Schneider, "Comparative stability of cloned and non-cloned code: An empirical study," in *Proc. ACM SAC*, 2012, pp. 1227–1234.

[27] E. Jürgens, F. Deissenboeck, M. Feilkas, B. Hummel, B. Schätz, S. Wagner, C. Domann, and J. Streit, "Can clone detection support quality assessments of requirements specifications?" in *Proc. ICSE*, 2010, pp. 79–88.

[28] C. K. Roy and J. R. Cordy, "A survey on software clone detection research," *School Of Computing TR 2007-541, Queen's University*, p. 115, 2007.

[29] B. Fluri and H. C. Gall, "Classifying change types for qualifying change couplings," in *Proc. ICPC*, 2006, pp. 35–45.

[30] A. Mockus and L. G. Votta, "Identifying reasons for software changes using historic databases," in *Proc. ICSM*, 2000, pp. 120–130.

[31] M. Mondal, C. K. Roy, and K. A. Schneider, "Identifying code clones having high possibilities of containing bugs," in *Proc. ICPC*, May 2017, pp. 99–109.

[32] L. Barbour, F. Khomh, and Y. Zou, "Late propagation in software clones," in *Proc. ICSM*, 2011, pp. 273–282.

[33] SourceForge. [Online]. Available: https:/sourceforge.net/

[34] ArtisticStyle. [Online]. Available: https://sourceforge.net/projects/astyle/

[35] Strike A Match. [Online]. Available: http://www.catalysoft.com/articles/strikeamatch.html

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

[37] T. Kamiya, S. Kusumoto, and K. Inoue, "CCFinder: A multilinguistic token-based code clone detection system for large scale source code," *IEEE TSE*, vol. 28, pp. 654–670, 2002.

[38] L. Jiang, G. Misherghi, Z. Su, and S. Glondu, "Deckard: Scalable and accurate tree-based detection of code clones," in *Proc. ICSE*, 2007, pp. 96–105.

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

[40] J. Svajlenko and C. K. Roy, "Cloneworks: A fast and flexible large-scale near-miss clone detection tool," in *Proc. ICSE-C*, 2017, pp. 177–179.

[41] Simian. [Online]. Available: http://www.redhillconsulting.com.au/products/simian/

[42] M. S. Uddin, C. K. Roy, K. A. Schneider, and A. Hindle, "On the effectiveness of simhash for detecting near-miss clones in large scale software systems," in *Proc. WCRE*, 2011, pp. 13–22.

[43] M. White, M. Tufano, C. Vendome, and D. Poshyvanyk, "Deep learning code fragments for code clone detection," in *Proc. ASE*, 2016, pp. 87–98.

[44] C. Roy and J. Cordy, "Nicad: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization," in *Proc. ICPC*, 2008, pp. 172–181.

[45] J. R. Cordy and C. K. Roy, "The nicad clone detector," in *Proc. ICPC*, 2011, pp. 219–220.

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

[47] Mann-whitney wilcoxon (mww) test. [Online]. Available: http://www.socscistatistics.com/tests/mannwhitney/Default2.aspx

[48] J. Cohen, *Statistical power analysis for the behavioral sciences (2nd ed.).*, 1988.

[49] K. O. R. McGraw and S. P. Wong, "Comparison and evaluation of code clone detection techniques and tools: A qualitative approach," *SCP*, vol. 11, pp. 470–495, 2009.

[50] J. F. Islam, M. Mondal, C. K. Roy, and K. A. Schneider, "A comparative study of software bugs in clone and non-clone code," in *Proc. SEKE*, 2017, pp. 436–443.

[51] T. Wang, M. Harman, Y. Jia, and J. Krinke, "Searching for better configurations: A rigorous approach to clone evaluation," in *Proc. FSE*, 2013, pp. 455–465.

[52] K. Inoue, Y. Higo, N. Yoshida, E. Choi, S. Kusumoto, K. Kim, W. Park, and E. Lee, "Experience of finding inconsistently-changed bugs in code clones of mobile software," in *Proc. IWSC*, 2012, pp. 94–95.

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

[54] H. Sajnani, V. Saini, and C. V. Lopes, "A comparative study of bug patterns in java cloned and non-cloned code," in *Proc. SCAM*, 2014, pp. 21–30.

[55] Experimental data. [Online]. Available: https://goo.gl/Xb8Cim