2011 19th IEEE International Conference on Program Comprehension

An Empirical Study of the Impacts of Clones in Software Maintenance

Manishankar Mondal Md. Saidur Rahman Ripon K. Saha
Chanchal K. Roy Jens Krinke* Kevin A. Schneider
University of Saskatchewan, Canada
{*mshankar.mondal, saeed.cs, ripon.saha, chanchal.roy, kevin.schneider*}*@usask.ca*
**University College London, UK*
j.krinke@ucl.ac.uk

***Abstract*—The impacts of clones on software maintenance is a long-lived debate on whether clones are beneficial or not. Some researchers argue that clones lead to additional changes during the maintenance phase and thus increase the overall maintenance effort. Moreover, they note that inconsistent changes to clones may introduce faults during evolution. On the other hand, other researchers argue that cloned code exhibits more stability than non-cloned code. Studies resulting in such contradictory outcomes may be a consequence of using different methodologies, using different clone detection tools, defining different impact assessment metrics, and evaluating different subject systems. In order to understand the conflicting results from the studies, we plan to conduct a comprehensive empirical study using a common framework incorporating nine existing methods that yielded mostly contradictory findings. Our research strategy involves implementing each of these methods using four clone detection tools and evaluating the methods on more than fifteen subject systems of different languages and of a diverse nature. We believe that our study will help eliminate tool and study biases to resolve conflicts regarding the impacts of clones on software maintenance.**

***Keywords*--Clone Evolution; Code Stability; Experiment.**

I. Introduction

Reuse of code fragments with or without modification by copying and pasting from one location to another is very common during software development. This results in the existence of the same or similar code blocks in different components of the software system. Code fragments that are exactly the same or are very similar to each other are known as clones. In addition to copy-paste activity, some other issues, including programmers' behaviour such as laziness and the tendency to repeat common solutions, technology limitations, code evolvability, code understandability and external business forces have influences on code cloning [5]. Whatever the causes of cloned code, the impacts of clones are of concern from a software maintenance point of view.

The question "Is Cloned Code Harmful?" has divided software engineering researchers into two main groups. One group, in favour of clones, concluded that clones are not harmful [6], [7], [15], [1], [2], instead clones can be helpful from different perspectives [5]. The second group identified clones as true "bad smells" and showed that clones have negative impacts on software quality and maintenance as cloning increases maintenance cost [12], [3], [11]. Moreover, clones can introduce faults during software maintenance and evolution if the cloned code is updated inconsistently [3].

If we observe from an analytical perspective, we can identify the possible candidate reasons behind these contradictory outcomes. First, different researchers have modelled and developed different impact evaluation systems using different clone detection tools. Second, they have evaluated their methods on different subject systems (code bases). Third, the impact evaluation metrics calculated in those systems were selected using different viewpoints.

In this paper we propose a uniform framework to evaluate the impacts of clones on software maintenance. To eliminate the existing contradictions among different studies, we propose an empirical study that incorporates nine leading existing methodologies [6], [7], [12], [15], [8], [2], [11], [9], [18] that analyzed the impacts of clones. Our research strategy involves the implementation of each of these methods using four clone detection tools including the hybrid clone detection tool, NiCad [13] and the evaluation of these methods on more than fifteen subject systems of diverse size, language and application domain. We expect that the integration of different methodologies within a common framework will eliminate the unintentional biases and our study will resolve the conflicts regarding the impact of clones in software maintenance.

The rest of the paper is organized as follows. Section II summarizes relevant research and outlines the background and context of the proposed research. Section III describes the proposed methodology in detail. The implementation of the proposed system is outlined in Section IV. Section V presents the decision making strategy and Section VI describes the experimental result of this research followed by the conclusion in Section VII.

II. Related Work

Recently, Hotta et al. [2] studied the impact of clones in software maintenance activities with a different approach where the modification frequencies of the duplicated and non-duplicated code segments were measured. Their implemented system works on different revisions of a subject system by automatically extracting the modified files across consecutive revisions. They conducted a fairly large study using different tools and subject systems which suggests that the presence of clones does not introduce extra difficulties to the maintenance phase.

Krinke [7] measured how consistently code clones are changed during maintenance using *Simian* [17] (a clone

1063-6897/11 $26.00 © 2011 IEEE
DOI 10.1109/ICPC.2011.14

detector) and *diff* (a file comparison utility) on Java, C and C++ code bases considering Type-I clones only. He found that clone groups changed consistently through half their lifetime. In another experiment he showed that cloned code is more stable than non-cloned code [8]. In his most recent investigation [9] centred on calculating the average ages of the cloned and non-cloned code, he has proved cloned code to be more stable than non-cloned code by exploiting the capabilities of a version controlling system.

Lozano and Wermelinger [12] experimented to assess the effects of clones on the changeability of software using CCFinder [4] as the clone detector. They have calculated three changeability measures: (i) likelihood; (ii) impact of a method change; and, (iii) work required for maintaining a method. According to their study, in at least 50% of the cases clones did not increase the changeability measures but sometimes these measures seemed to increase for the part of the systems related to the cloned methods. In another experiment [10], they experienced that cloned code leads to more changes. In their most recent experiment [11] aiming to analyze the imprints of clones over time, they calculated the extension of cloning, and measured the persistence and stability of cloned methods by improving their previous studies. Their study suggests that cloned methods remain cloned most of their lifetime and that cloning introduces a higher density of modification during maintenance.

Kim et al. [6] proposed a model of clone genealogy to study clone evolution. Their study with the revisions of two medium sized Java systems showed that refactoring of clones may not always improve software quality. They also argued that aggressive and immediate refactoring of short-lived clones is not required and that such clones might not be harmful. Saha et al. [15] extended their work by extracting and evaluating code clone genealogies at the release level of 17 open source systems involving four different languages. Their study reports similar findings to Kim et al. and concludes that most of the clones do not require any refactoring efforts in the maintenance phase. On the other hand, Juergens et al.'s [3] study with large scale commercial systems suggests that inconsistent changes are very frequent in cloned code and nearly every second unintentional inconsistent change to a clone leads to a fault.

Kapser and Godfrey [5] identified different patterns of cloning and experienced that around 71% of the clones could be considered to have a positive impact on the maintainability of the software system.

Aversano et al. [1] combined clone detection and modification transactions on open source software repositories to investigate how clones are maintained during evolution and bug fixing. Their study reports that most of the cloned code is consistently maintained. In another similar but extended study, Thummalapenta et al. [18] indicated that in most of the cases clones are changed consistently and for the remaining inconsistently changed cases, clones mainly undergo independent evolution.

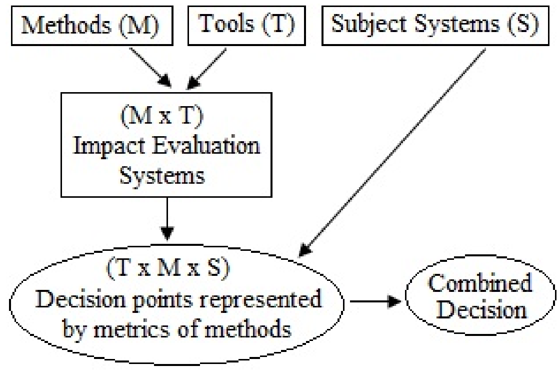

Figure 1. The proposed decision making procedure

We see that while the objective is the same, *determining the impacts of clones on software maintenance*, researchers considered different approaches with different clone detection tools and different subject systems, which may be the reason there are contradictory findings. It would be interesting to see if these contradictions can be resolved.

III. PROPOSED METHODOLOGY

From the implementation point of view, differences among previously developed systems occurred in four dimensions. These are (a) underlying methodologies, (b) clone detection tools, (c) metrics, and (d) subject systems.

The metrics used by the methods are different but the intention is to assess the impact of clones on maintenance activities. In our empirical study, we plan to implement nine existing leading methods using four clone detection tools following the decision making architecture in Fig. 1.

The differences in the dimensions (b) and (d) will be eliminated by using the same clone detection tools for each of the systems and applying the same subject systems (code bases) to them.

According to the architecture, our implementation strategy consists of fours steps: 1) selection of methodologies to implement,; 2) selection of clone detection tools to implement the methodologies; 3) selection of metrics to calculate; and, 4) selection of subject systems to evaluate the implemented methodologies.

A. Methodology Selection

We plan to incorporate almost all of the existing but leading studies that reported contradictory findings. Table I represents the nine methods that we have selected for our study. Among these, two methods proposed by Lozano and Wermelinger [12], [11] assessed cloning as harmful for maintenance while the remaining methods concluded the opposite.

B. Tool Selection

In order to avoid further bias of clone detection tools, we plan to use four clone detection tools including the hybrid clone detection tool NiCad [13] that has been shown to give

Table I
LIST OF METHODS TO INCORPORATE IN OUR FRAMEWORK

Incorporated Methods	Supported Language	Tools Used	Clone Gran.	Clone Types
Krinke [7]	Java, C++, C	Simian, diff	AB	1
Lozano and Wermelinger [12]	Java	CCFinder, CTAGS	M	1
Lozano and Wermelinger [11]	Java	CCFinder	M	1, 2
Hotta et al. [2]	Java, C++, C	CCFinder(X), Simian, Scorpio	AB	1, 2, 3
Kim et al. [6]	Java	CCFinder, diff	AB	1
Saha et al. [15]	Java, C++, C#, C	CCFinder	AB	1, 2
Krinke [9]	Java	Simian	AB	1
Thummalapenta et al. [18]	Java, C	CCFinder, SimScan, ccdiml.	AB	1, 2, 3
Krinke[8]	Java, C++, C	Simian, diff	AB	1

AB: Arbitrary Block, M: Method

Table II
LIST OF SUBJECT CLONE DETECTION TOOLS

Subject Tools	Detection Approach	Supported Language	Clone Types
CCFinderX [4]	Token	Java, C++, C, Cobol, C# etc.	1, 2
NiCad [13]	Text/Parser	Java, C, C#, Python	1, 2, 3
Simian [17]	Text	Java, C++, C, C# etc.	1
Scorpio [16]	PDG	Java	1, 2, 3

both high precision and recall [14]. The subject tools that we plan to use are listed in Table II.

C. Selection of Metrics

We plan to consider all the major metrics of the nine studies listed in Table III. We believe that this comprehensive set of metrics with four state-of-the-art clone detection tools and more than 15 subject systems will provide us with fairly unbiased results on the impacts of clones on software maintenance and at the same time will help us find important insights on the evolution of clones and their management.

D. Subject Code Bases / Input Selection

In order to avoid the sampling bias (at least partially) we plan to use more than 15 subject systems of diverse varieties and different languages for each of the nine methods (and their metrics) selected. Our selection strategy will include some of the previously studied systems along with several new systems that have not been studied previously. We will work on many revisions (and releases where applicable) of these code bases tracked by SVN.

IV. IMPLEMENTATION OUTLINE

Implementation of the selected methods using the selected tools and decision making by applying the methods to the selected subject systems will be done in the following steps.

A. Step 1: Collecting the repositories

We use SVN to collect the subject systems from open source software repositories.

Table III
LIST OF METRICS TO BE EVALUATED

Methods proposed by	Selected Metrics to be Considered
Krinke [7]	(i) Proportion of consistently changed clone groups and (ii) Proportion of inconsistently changed clone groups
Lozano and Wermelinger [12]	Likelihood and impact for changing a method and work required for corresponding method change.
Lozano and Wermelinger [11]	Extension (Proportion of tokens in cloned region of a method), persistence (percentage of cloned life time of a method) and stability per method and per application.
Hotta et al. [2]	Modification frequencies of cloned and non-cloned code for a range of revisions of a specific subject system.
Kim et al. [6]	(i) Number of genealogies changed consistently, (ii) Number of locally unfactorable genealogies, (iii) Proportion of long lived genealogies and (iv) Proportion of k-volatile genealogies.
Saha et al. [15]	(i) Proportion of consistently changed genealogies, (ii) Proportion of syntactically similar genealogies and (iii) Proportion of alive and dead genealogies.
Krinke [9]	(i) Average last change date of cloned code in a file (ii) Average last change date of non-cloned code in a file and (iii) Proportion of files where cloned code is older than non-cloned code.
Thummalapenta et al. [18]	Normalized Levenshtein Distances (NLD) for clone section pairs will be calculated to investigate the clone evolution pattern.
Krinke [8]	Stability of cloned and non-cloned code with respect to addition, deletion and change.

B. Step 2: Implementing the selected methods

The selected methods are implemented using Java. Moreover, as we will use the previously mentioned four tools to implement each of the methods, there will remain four working copies of each of them after we have completed the implementations. Thus, for the 9 selected methods we will have 36 working systems in total.

C. Step 3: Analyzing metrics to make a combined decision

We apply each of the working systems on each of the subject systems to make the final decision (Section V).

V. OUR PROPOSED DECISION MAKING STRATEGY

For each combination of tools and subject systems, we will run the selected nine methods and generate their corresponding metrics. While we will observe and analyse the data for each of the individual runs, the overall decision will be made in the following way.

Let the number of methods $= m$
Methods are denoted by M_i where $1 \leq i \leq m$
Let the number of tools used $= t$
Tools are denoted by T_j where $1 \leq j \leq t$
Let the number of subject systems $= c$
Subject Systems/Code Bases are denoted by C_k where $1 \leq k \leq c$
From each combination of tools and subject systems the decision made for a method is denoted D_{ijk}
Where i is the index of the method

Table IV
RESULTS FOR HOTTA ET AL. [2] AND KRINKE [9]

System/Method		Hotta et al. [2]		Krinke [9]		
Subject System	Up to i_{th} Revision	MF of CC	MF of non-CC	Files with CC	% of files where CC is older than non-CC	% of files where CC is younger than non-CC
ThreeCAM	14	7.64	4.02	12	25.00	16.67
DatabaseToUML	60	11.05	47.28	57	1.75	7.02
jEdit	150	1.44	4.45	317	9.46	44.16
NatMonitor	88	5.09	3.35	2	0.00	100.00
OpenYMSG	100	10.32	9.04	14	24.00	16.67

MF: Modification Frequency, CC: Cloned Code

j is the index of the selected tool
k is the index of the selected code base.

For each method M_i, we get $(t \times c)$ sets of the metrics defined for M_i. As a result, for each M_i we will have a total of $(t \times c)$ decisions. From these decisions we will calculate a combined decision CombinedDecision (M_i) for method M_i.

$$CombinedDecision(M_i) = \cup_{j=1}^{t} \cup_{k=1}^{c} D_{ijk} \tag{1}$$

Thus, from m methods we will get m combined decisions. From these m decisions we will possibly get an unbiased overall decision about the impact of clones on software maintenance. The overall decision is expressed as follows.

$$OverallDecision(m) = \cup_{i=1}^{m} CombinedDecision(M_i) \tag{2}$$

The unit values of D will be between -1 to +1 depending on whether a particular metric of a particular method indicates a negative or positive impact of clones on software maintenance for a particular subject system with a particular subject clone detection tool. It remains for us to determine how the results of the different methods will be mapped to this range.

VI. EXPERIMENTAL RESULTS

So far we have implemented three methods proposed by (i) Hotta et al. [2], (ii) Krinke [9] and (iii) Lozano and Wermelinger [12] using SVN repositories and have obtained results for the first two methods. The results are listed in Table IV for five subject systems where we use CCFinderX for clone detection. Here the minimum length of the clones for CCFinderX was specified as 50 tokens.

Results in the second column of Table IV comply with those reported by Hotta et al. [2]. Out of five examined subject systems, two exhibit lower modification frequencies for cloned code than non-cloned code where the remaining three yields the opposite results.

The results in the last column of Table IV were obtained with Krinke's method [9] using the same five subject systems from which only jEdit was used in the original study [9], however, we used the 150th revision, whereas in the original study he used the 19285th revision. Moreover, we used CCFinderX while he used Simian. For three out of the five subject systems, the calculated metrics' values do not suggest the stability of cloned code over non-cloned code.

Most striking is that for four out of the five systems, the two methods have contradicting results: For two systems, Krinke's method [9] shows the stability of cloned over non-cloned code with a greater proportion of files where cloned code is older than non-cloned code. On the other hand, the method proposed by Hotta et al. [2] suggests a higher stability of non-cloned code over cloned code by showing a lower modification frequency for the non-cloned code. For two other systems, it is the other way round. Only for the smallest system (NatMonitor) do both methods agree.

VII. CONCLUSIONS

In conclusion, it can be argued that our empirical study will resolve a long lived debate about the impact of clones on software maintenance. From this study we expect to draw a firm decision on whether cloning is harmful for software maintenance, and if so to what extent. We then plan to determine what could be done to overcome the harmful effects (if any) of clones during maintenance including a novel proposal for a clone management system.

Acknowledgments: This work is supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

REFERENCES

[1] L. Aversano, L. Cerulo, and M. D. Penta, "How clones are maintained: An empirical study," Proc. *CSMR*, 2007, pp. 81–90.
[2] K. Hotta, Y. Sano, Y. Higo, S. Kusumoto, "Is Duplicate Code More Frequently Modified than Non-duplicate Code in Software Evolution?: An Empirical Study on Open Source Software," Proc. *EVOL/IWPSE*, 2010, pp. 73–82.
[3] E. Juergens, F. Deissenboeck, B. Hummel, S. Wagner, "Do Code Clones Matter?," Proc. *ICSE*, 2009, pp. 485– 495.
[4] T. Kamiya, S. Kusumoto, and K. Inoue, "CCFinder: a multilinguistic token-based code clone detection system for large scale source code," TSE, 28(7), 2002, pp. 654–670.
[5] C. Kapser and M. W. Godfrey, ""Cloning considered harmful" considered harmful: patterns of cloning in software," Emp. SE. 13(6), 2008, pp. 645–692.
[6] M. Kim, V. Sazawal, D. Notkin, and G. C. Murphy, "An empirical study of code clone genealogies," Proc. *ESEC-FSE*, 2005, pp. 187–196.
[7] J. Krinke, "A study of consistent and inconsistent changes to code clones," Proc. *WCRE*, 2007, pp. 170–178.
[8] J. Krinke, "Is cloned code more stable than non-cloned code?," Proc. *SCAM*, 2008, pp. 57–66.
[9] J. Krinke, "Is Cloned Code older than Non-Cloned Code?," Proc. *IWSC*, 2011, 7 pp. (to appear)
[10] A. Lozano, M. Wermelinger, and B. Nuseibeh, "Evaluating the Harmfulness of Cloning: A Change Based Experiment," Proc. *MSR*, 2007, pp. 18.
[11] A. Lozano and M. Wermelinger, "Tracking clones imprint," Proc. *IWSC*, 2010, pp. 65–72.
[12] A. Lozano, and M. Wermelinger, "Assessing the effect of clones on changeability," Proc. em ICSM, 2008, pp. 227–236.
[13] C.K. Roy and J.R. Cordy, "NICAD: Accurate Detection of Near-Miss Intentional Clones Using Flexible Pretty-Printing and Code Normalization," Proc *ICPC*, 2008, pp. 172–181.
[14] C. K. Roy and J. R. Cordy, "A mutation / injection-based automatic framework for evaluating code clone detection tools," Proc. *Mutation*, 2009, pp. 157–166.
[15] R. K. Saha, M. Asaduzzaman, M. F. Zibran, C. K. Roy, and K. A. Schneider, "Evaluating code clone genealogies at release level: An empirical study," Proc. *SCAM*, 2010, pp. 87–96.
[16] Scorpio. http://www-sdl.ist.osaka-u.ac.jp/~higo/cgi-bin/moin.cgi/scorpio-e/.
[17] Simian-similarity analyser. http://www.redhillconsulting.com.au/products/simian/
[18] S. Thummalapenta, L. Cerulo, L. Aversano, and M. D. Penta, "An empirical study on the maintenance of source code clones," *ESE*, 15(1), 2009, pp. 1–34.