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
{manishankar.mondal, saeid.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
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

**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
A. Step 1: Collecting the repositories

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

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).

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 I
LIST OF METHODS TO INCORPORATE IN OUR FRAMEWORK

<table>
<thead>
<tr>
<th>Incorporated Methods</th>
<th>Supported Language</th>
<th>Tools Used</th>
<th>Clone Gran.</th>
<th>Clone Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Krinke [7]</td>
<td>Java, C++, C</td>
<td>Simian, diff</td>
<td>AB</td>
<td>1</td>
</tr>
<tr>
<td>Lozano and Wer-</td>
<td>Java</td>
<td>CCFinder,</td>
<td>M</td>
<td>1</td>
</tr>
<tr>
<td>Lozano and Wer-</td>
<td>Java</td>
<td>CCFinder</td>
<td>M, 1, 2</td>
<td></td>
</tr>
<tr>
<td>melinger [11]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hotta et al. [2]</td>
<td>Java, C++, C</td>
<td>CCFinder(X), Simian, Scorpio</td>
<td>AB, 1, 2, 3</td>
<td></td>
</tr>
<tr>
<td>Kim et al. [6]</td>
<td>Java</td>
<td>CCFinder,</td>
<td>AB</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>diff</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Saha et al. [15]</td>
<td>Java, C++, C#</td>
<td>CCFinder</td>
<td>AB</td>
<td>1, 2</td>
</tr>
<tr>
<td>Melinger [8]</td>
<td>Java, C++, C</td>
<td>Simian, diff</td>
<td>AB</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table II
LIST OF SUBJECT CLONE DETECTION TOOLS

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

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.

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
From which only jEdit was used in the original study [9], with Krinke’s method [9] using the same five subject systems three yields the opposite results. Those reported by Hotta et al. [2]. Out of five examined Table IV for five subject systems where we use CCFinderX (i) Hotta et al. [2], (ii) Krinke [9] and (iii) Lozano and this range.

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.

\[
OverallDecision(m) = \bigcup_{i=1}^{m} CombinedDecision(M_i)
\]

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