Measuring the Efficacy of Code Clone Information: An Empirical Study

Debarshi Chatterji[1], Beverly Massengill[2], Jason Oslin[1], Jeffrey C. Carver[1], Nicholas A. Kraft[1]

[1]University of Alabama
{dchatterji, oslin002}@crimson.ua.edu; {carver, nkraft}@cs.ua.edu

[2]Tennessee Tech University
bamassengi21@tntech.edu

Abstract

Much recent research effort has been devoted to designing efficient code clone detection algorithms and tools. However, there has been little human-based empirical study of how the output of those tools is used by developers when performing maintenance tasks. In this study 43 computer science graduate students completed a bug localization task in which a clone was present while researchers observed their activities. The goal of the study was to understand how those developers use clone information to perform this task. The results of this study showed that participants who used the clone information correctly, i.e. they first identified a defect then used it to look for clones of the defect, were more effective than participants who used the information incorrectly. The results also showed that participants who had industrial experience were more effective than those without industrial experience.

General Terms Measurement, Performance, Correctness, Human Factors, Experimentation.

Keywords software clones; empirical studies; software engineering; code clone report; software maintenance

1. Introduction

A common approach employed by developers when coding is to identify sections of code that can be copied, pasted and possibly edited, rather than writing completely new code [9]. This copied code is referred to as a *code clone*. There are many reasons why a developer may choose to clone code rather than writing new code, e.g. they think it is faster. On average, between five and twenty percent of software systems are constituted of cloned code [16].

There are conflicting views on whether code clones are problematic. Earlier work by Fowler et al. classified code clones as a 'bad smell' that would increase the difficulty of maintenance [5]. However, more recent work indicates that code clones may not be as harmful as originally believed [9,15]. In fact, code clones may improve productivity [7]. Rahman et al. found little empirical evidence that clones negatively affect software maintainability but did find that cloned code may be less fault prone than non-cloned code [12]. However, everyone agrees that awareness of code clones is critical to support correct (and complete) performance of software maintenance tasks. Consequently, in recent years, researchers have been developing approaches for clone detection in source code, along with accompanying tools to support those approaches.

It appears to be assumed by the creators of these approaches, that if a developer is provided with code clone information, they will know how to use it. We have seen no studies that actually verify the usefulness of the presented code clone information. Therefore, the main motivation behind this study was observe how developers would use a 'code clone report' while performing a bug localization task. We wanted to understand, given a clone report and no instruction on using the information contained in the report, how a developer would utilize the report. We developed the following research questions to drive the study:

- RQ 1*: How do developers use a clone information report in bug localization task?*
  - RQ 1.1*: Will developers use the clone information report?*
  - RQ 1.2*: Does the clone information aid developers in successfully identifying where multiple changes must be made?*
  - RQ 1.3*: Does the clone information lead developers to identify false-positives?*
- RQ 2*: Do novice and professional developers use the clone information differently? If so, how?*

Based on how the developers make use of the clone report, the long term goal of our work is to suggest ways to improve the code clone reports to make developer more efficient when using them.

The remainder of the paper is organized as follows: Section 2 describes the related work. Section 3 provides details about the study. Section 4 contains a detailed data

This paper was published in the proceedings of the Workshop on Evaluation and Usability of Programming Languages and Tools (PLATEAU) at the ACM Onward! and SPLASH Conferences.
October, 2010. Reno/Tahoe, Nevada, USA..

analysis. Section 5 describes the threats to validity. Section 6 contains the conclusions and plans for future work.

2. Related work

There has been a considerable amount of research conducted on clone detection approaches and tools. Bellon, et al., give a detailed description of the types of clone detection tools and the output produced by these tools [2]. This paper was helpful in understanding how the clone report is generated by various tools, specifically CCFinder [6], which was used in our study.

Unfortunately, there are very few empirical studies that focus on understanding how software developers use the output of such approaches to maintain code. The most relevant empirical study to our research, by Kim et al., focused on the coding habits of professional software developers. The study provides insight into the positive and negative aspects of copy and paste behavior. The researchers began by using naturalistic observation and later shifted to automated data collection, when they noticed that the observers were creating an unnecessary disturbance to the participants [9]. In our study, we collected data through the same type of naturalistic observation, without use of automated data collection. In our case, the observers did not interfere with or disturb the activities of the participants. In addition, the observers were able to answer the participants' questions about the conduct of the study. We considered adapting automated data collection. Because this approach has its own share of shortcomings, we chose not to use it in this study.

Some previous work provides some insight into the potential impacts that code clones have. Krinke, et al.'s study of open source software indicated that half of the changes to code clone groups are inconsistent with each other [11]. A study by Cordy, et al. reports that removing clones actually increases risk in large software systems [3]. Similarly, Kapser, et al. claim that clones have a positive impact on maintainability. They describe several patterns of cloning and discuss their benefits for the long term evolution of software Instead of eradicating repeated code there should be effort towards developing tools to support long term maintenance of clones [7].

3. Study Overview

The goal of this study was to understand how developers use the output of a code clone detection tool while performing a standard maintenance task of bug localization. To make the study tractable, we used bug localization (i.e. identifying the source code elements that must be modified to correct a bug) rather than actual bug repair. This choice seems reasonable as the main use of clone information is to help a developer identify cloned code that must change, rather than to actually make the change. In order to conduct this study, we chose a software application that met some specific requirements. Section 3.1 describes the software application used in the study. Section 3.2 then gives a detailed description of the study design. Finally, Section 3.3 explains the data that was collected during the study.

3.1 Software System: Apache Ant 1.6.5

Because the difficulty of the bug localization task was highly dependent upon the software system being examined, we determined that the software should be:

- Large enough to make bug identification a nontrivial task;
- Small enough that participants could perform the bug localization task in one hour; and
- Contained both a bug that was in cloned code and a bug that was in non-cloned code to control for the placebo effect.

To meet these requirements we chose *Apache Ant* version 1.6.5. Apache Ant automates the software build process, similar to Make. We chose version 1.6.5 because it contained two distinct bugs that met our requirements. Bug 1 had two instantiations, or clones, in different methods within the same class. Bug 2 was a simpler bug and had only one instantiation (i.e. no clones). So, each participant had the opportunity to identify three correct defects (both instantiations of bug 1 and the one instantiation of bug 2) and one incorrect report of a false positive (a second instantiation of bug 2 which did not exist). The two bugs are user-reported bugs available in the Apache Ant Bugzilla repository. We relied on the posted solution in Bugzilla to verify the correctness of the participants. Due to the presence of solutions on the web, the participants were not allowed access to the internet during the study.

3.2 Study Design

The Training: The course instructors gave the participants a brief lecture about code clones and their impact on software development and maintenance. These lectures familiarized the participants with the terms *code clone* and *code clone report*. The lectures did not give the participants specific guidance on how to use the clone information, because we were testing the efficacy of the clone report for the typical developer, not one with detailed knowledge about code clones and the use of code clone tools.

After the training, the participants received an overview document containing the following information needed to complete the experimental tasks.

1. A brief introduction to code clones, the Apache Ant system and definitions of important terms.
2. A directory structure of Apache Ant 1.6.5.
3. Information on how to reproduce the two bugs.

The participants also received the Ant documentation, the j2dsk-1.4.2 documentation, and the clone report.

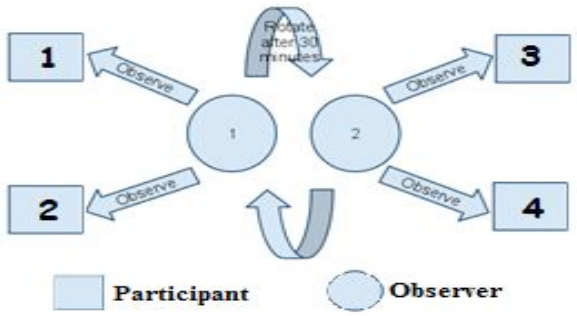

Figure 1. Study Design

The Sample: The 43 participants were drawn from graduate software courses, 13 from the University of Alabama (UA) and 30 from the University of Alabama in Huntsville (UAH). As discussed in the results section, some participants were novices (little or no industrial experience) while others had industrial experience.

The Task: The participants performed the bug localization task for the two Apache Ant 1.6.5 bugs described in the overview documents. Based on the information provided during training, the participants were instructed to identify the specific location (file, method and line number) that should be modified to correct the bug. They were told that they should identify all portions of the code that must be modified to correct the bug. The participants were not expected to actually make the repair. All tasks were conducted in a Linux environment. The participants were allowed to use the standard Linux utilities "find" and "grep" within the terminal, or were allowed to use the search dialog box provided by Ubuntu. The participants were not provided with any means to execute the code. Once they had identified the bug, they recorded their findings in the *defect identification sheet* (Section 3.3).

3.3 Data Collection

We used two types of data collection in this study: naturalistic observation and self-reported data.

Naturalistic Observation While performing the bug localization tasks, two of the authors acted as passive observers. The observers stood behind the participants and watched over their shoulder as they worked. Because the participants performed the study in groups of four, the two observers divided their time observing the four participants. To perform this division of time, each observer was responsible for two participants. They alternated between observing the participants every two minutes. After 30 minutes (half of the experimental session), the observers switched and observed the two participants whom they had not observed during the first 30 minutes.

In cases where there were less than four participants in a session, the observers observed the participants who were present for two minutes, then did not observe them for two minutes to ensure that the same data was collected throughout the study. Figure 1 shows the layout of the room in which the experiment was conducted.

The observers maintained a 'fly on the wall' profile unless the participants specifically asked them questions. The participants asked questions like: "What exactly are we supposed to do here?"; "How do I read the clone report?"; "Can I execute the code?"; "Can I access the internet?"; and "I am not able to find the bug, what should I do?". The observers kept their answers as brief as possible so as not to give away any information that may have biased the study or helped the participant complete the task.

During the observations, the observers recorded information about which resources the participant was using (i.e. the code, the code clone report, the report form and the overview sheet), what actions the participant was performing, and subjective notes about the participant's actions. To record this information, the observers used the *Observer's Data Recording Form* shown in Figure 2.

Session # _____

Time: _____ Subject#: ____	**Activity**: ☐familiarizing ☐reporting **Resource**: ☐clone info ☐code ☐documentation **Notes**: _____
Time: _____ Subject#: ____	**Activity**: ☐familiarizing ☐reporting **Resource**: ☐clone info ☐code ☐documentation **Notes**: _____

Figure 2: Observers' data recording form

Self-Reported Data To collect self-reported data, we used two different data collection forms: the *bug identification form* and the *post observation questionnaire*. The participants used the *bug identification form* (Figure 3) to report the bugs that they found in the system. The form includes places to record a time stamp, bug location, description of defect, steps taken to locate the defect and how they would fix the defect. The defect identification sheet has space to report four defects, two for each bug (the original defect and the clone if present). Even though only one bug had a clone, so as not to bias the results, the defect form provided space to an original defect and a clone for each bug. The information from this form helped us calculate the number of successful defects detected, false positives if any and the time they took to find the bugs.

The *post-observation questionnaire* was given to the participants at the end of the experiment. On it the participants reported their experience with software development experience, bug localization, code clones and CCFinder (the tool whose output was provided to the participants as the clone report). The participants were also

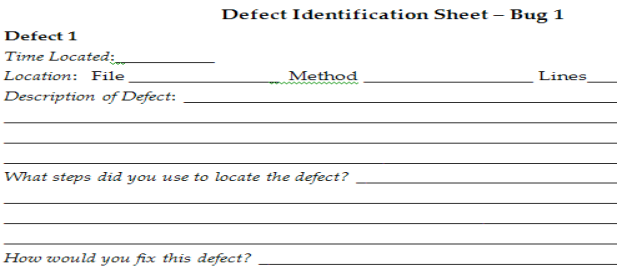

Figure 3: Bug Identification Form

provided with space to give their opinion of the usefulness of the clone report for finding and locating the defects.

4. Data Analysis

This section is organized around the research questions presented in Section 1. We use an alpha value of 0.05 for judging statistical significance.

4.1 Overview of Data

Figure 4 partitions the sample based their performance in terms of finding the three true defects and the one false positive. The largest group of participants was those who found none of the defects. Prior to this study, most participants had little experience identifying defects in large software systems. The data showed that participants who had some software development experience, as reported in the *post observation questionnaire*, were more confident and more efficient performing the task. The small group of participants who found all three defects had all worked on multiple industrial software projects. Another observation from this data is that in spite of having a detailed clone report; several participants were able to locate the initial defects but were not able to find the clone defect for Bug 1. In addition, some participants found a false positive (the nonexistent clone for Bug 2).

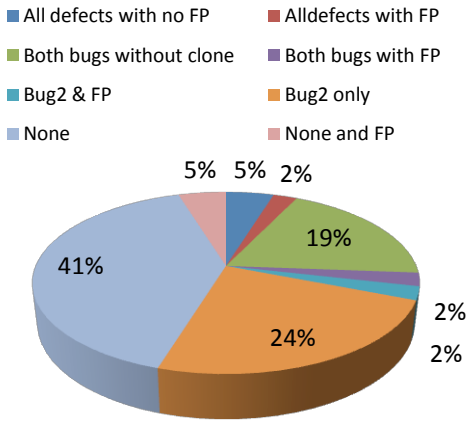

Figure 4: Partition of Sample (FP=False Positives)

4.2 RQ 1.1 – Will Developers use Clone Information

Research Question 1.1 focused on understanding whether the participants would use the clone information at all, and if so, how they would use it. On the *post observation questionnaire* the participants reported the efficacy of the clone report for the bug localization task on a five-point Likert scale (1-not effective at all to 5-highly effective). We grouped the participants into those that had a positive impression of the clone report (4 and 5 on the scale) and those that had a negative impression (3 or lower). We then performed a t-test between these two groups to compare the number of defects found (two defects for bug 1 and one defect for bug 2). The participants who had a positive impression of the clone report were more effective (1.45 defects vs. .78 defects on average) suggesting that people who find the clone report useful report more defects. This difference was not significant (t_{41} = 1.946; p = .059). To gain more insight into these results, the 19 participants who did not find any defects were excluded and the same analysis performed. Those who found the clone report more useful were significantly more effective (2.29 defects vs. 1.47 defects on average) (t_{22} = 2.739; p =.012).

At this point, we cannot claim causality between these two variables. We are not sure whether 1) finding the clone report useful caused the increase in defects found 2) finding more defects resulted in a more positive view of the clone report, or 3) some other variable influenced the results.

In order to evaluate how the clone report was being used, we had to define *correct* usage. If a participant fully understood how to use the clone report, he would realize that the clone report was only useful after identifying a defect to help search for clones of the problematic code. Because the clone report is merely a list of cloned code, it is not useful in identifying the original defect. Therefore, we defined *correct* usage of the clone report as using the clone report to search for clone code only after first identifying a defect. We defined *incorrect* usage as usage of the clone report prior to identifying a defect. Using these definitions, we characterized each participant as either correct or incorrect. However, because many participants were unfamiliar with a clone report, during the first ten minutes a participant could familiarize himself with the clone report without being characterized as incorrect.

Most participants did not use the clone report correctly - only eight participants used the correct process. Many of the participants appeared to be attempting to use the clone report to locate the initial defects, which would suggest that the clone report requires at least a small amount of training before being used for maximum efficacy.

4.3 RQ 1.2 – Is Clone Information Useful

Research Question 1.2 focused on understanding whether the clone information helps developers find defects. In

order to conduct this analysis we divided the sample into two groups based on the definition of *correct* and *incorrect* usage of the clone information as defined in Section 4.1. Then, we conducted two analyses.

First, we tested whether participants who used the correct process were more effective overall than those who used the incorrect process. Figure 5 shows that those who used the correct process found an average of 1.62 defects compared with 0.8 defects (out of a possible 3) for those who used the incorrect process. An independent samples t-test showed a significant difference (t_{41} = -2.146; p=.038).

Second, we tested whether the participants who used the correct process were more effective in finding the cloned defect related to Bug 1. However, this difference was not significant (t_{41} = -1.713; p = .094).

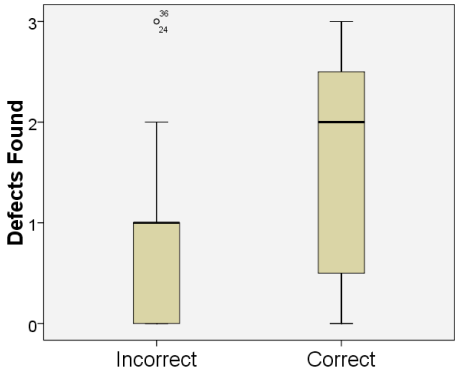

Figure 5: Correct vs. Incorrect Use of Clone Report

4.4 RQ 1.3 – Does Clone Information Lead to False Positives

Research Question 1.3 focused on understanding whether or not the clone information report misleads a developer towards detection of false positives while looking for the clone of a bug. None of the participants who used the clone report correctly reported a false positive. All five participants who did identify a false positive used the clone report incorrectly. While this result was not significant (likely due to small number of participants who used the correct process), the fact that no one that used the correct process reported a false positive is an encouraging result

4.5 RQ 2 – Novices vs. Professionals

To determine the effects of the participants' previous programming experience, we divided the sample into two groups: novices and professionals. On the *post observation questionnaire,* participants indicated their experience on a five-point Likert scale. Participants who answered 1, 2 or 3 (indicating either no experience or only classroom experience) were judged to be *novices*. Those who answered 4 or 5 (indicating experience in industry on one or more than one project) were judged to be *professionals*.

Using a similar analysis as in RQ1.2, we compared the effectiveness of novices and professionals. On average, the professionals identified more defects than the novices (1.53 vs. .43 respectively). This difference was significant (t_{41} = -4.222; p < .001). In addition, all four participants who correctly identified the cloned defect were professionals.

The professionals also tended to use the clone report correctly more often than the novices did. Table 1 is the contingency table showing the distributions of these two variables. Even though a chi-square test did not show a significant result (X^2 = 1.010; p= .315), there is a positive trend suggesting that the experienced participants tend to use the clone report correctly more often than novices.

These results suggest that clone detection ability is heavily reliant on experience, as professionals tended to use the clone report correctly more often than novices. However, this result could also have been caused by a another factor: the professionals identified significantly more defects than the novices, so the novices had a lower probability of finding the clone defect because finding the initial defect is a prerequisite to finding the cloned defect.

Count

	Use of Clone Report		
	incorrect	correct	Total
Novice	20	3	23
Professional	15	5	20
Total	35	8	43

Table 1: Novice/Professional vs Correct/Incorrect

5. Threats to Validity

Construct Validity: We defined the correct and incorrect usage of clone information based on theory and on participant observation. It is possible that this definition is incorrect or that the method we used to partition the participants into the groups was biased. Either of these problems could introduce a threat to construct validity.

Internal Validity: We chose a system written in JAVA because we assumed most students would be familiar with that language. If someone was not familiar with JAVA, it could bias the results. Ideally the participants should have had an option to select the programming language with which they were most familiar. The participants performed the task in a Linux environment which could be a validity threat if they were not familiar with Linux and the search features it provides. Finally, to prevent a threat to validity of the participants being able to locate the solutions on the web, we did not provide internet access. This choice could also introduce a threat in that it changes the typical working environment for a developer.

External Validity: The participant population consisted of both novices and professionals, reducing a threat to

external validity. Conversely, the task was performed in isolation rather that as part of a complete maintenance process, which may introduce a threat to external validity. The task was a realistic task (i.e. bug localization), but it was not complete (i.e. the participants did not fix the bug). Finally, the participants were not trained in using the clone report. Although this lack of training could introduce a threat to validity, one of our study goals was to see how people would use the report without training.

6. Conclusion and future work:

There have been few human-based empirical studies focused on code clones and clone reports. This exploratory study provides insights into how developers use a clone report. Some concrete conclusions from the study are:

- Initial evidence shows that, without training, most participants used the clone report incorrectly. Some argue that clone detection tools are needed to support the maintenance process because those tasks are often assigned to entry level developers. However, if those developers cannot use the clone report correctly, such clone detection tools are of little use to them.
- There is a relationship between correct use of the clone report and effectiveness, although we have yet to not establish a causal relationship.
- In a large software system the clone report might not help developers locate the initial defect, but it will help them locate clones of that defect. Use of the clone report may also reduce reporting of false positives.

As this was an exploratory study, there are a number of replications that can be performed. Since the lack of training on how to use clone information was a threat to validity, a replication can be done to control for clone detection training (i.e. by training half of the participants how to use the clone information correctly and not training the other half). Another replication can be performed in an environment where the participants could execute the code and repair the defects to see whether the results differ. Another replication will look at defects involving more than two instances of cloned code. For example, assume there are eight clones, but only three cause defects.

We will also investigate the false positives in more detail. We will look at the effects of the clone report on false positive identifications For example, if someone incorrectly identifies a defect and then, using the clone report, do they identify additional false positives?

Finally, this study serves as a starting point for a series of studies in which we will evaluate the effectiveness of various types of clone detection tools as well as various methods for presenting clone information to developers.

7. Acknowledgements

We thank Dr. Letha Etzkorn for allowing us to run the study in her class at UAH. We also thank the participants for their time to complete this study. We acknowledge support from NSF grants CCF-0915559 and CCF-0851824.

8. References

[1] B. S. Baker. (2007, Finding clones with dup: Analysis of an experiment. *IEEE Trans. Software Eng. 33*pp. 608-621.

[2] S. Bellon, R. Koschke, G. Antoniol, J. Krinke and E. Merlo. (2007, Comparison and evaluation of clone detection tools. *IEEE Trans. Software Eng. 33*pp. 577-591. Available:

[3] J. R. Cordy, T. R. Dean and N. Synytskyy. Practical language-independent detection of near-miss clones. Presented at CASCON '04: Proceedings of the 2004 Conference of the Centre for Advanced Studies on Collaborative Research.

[4] R. Falke, P. Frenzel and R. Koschke. (2008, Empirical evaluation of clone detection using syntax suffix trees. *Empirical Softw.Engg. 13(6),* pp. 601-643.

[5] Fowler M, Beck K, Brant J, OpdykeW, Roberts D. Refactoring: Improving the Design of Existing Code. 1st edn., Addison-Wesley, 1999.

[6] T. Kamiya, S. Kusumoto and K. Inoue. (2002, jul). CCFinder: A multilinguistic token-based code clone detection system for large scale source code. *IEEE Trans. Software Eng. 28(7),* pp. 654-670.

[7] C. Kapser and M. W. Godfrey. Cloning considered harmful considered harmful. Presented at WCRE '06: Proceedings of the 13th Working Conference on Reverse Engineering.

[8] Kapser C, Godfrey M. "cloning considered harmful" considered harmful. Proceedings of the 13th Working Conference on Reverse Engineering, Benevento, Italy, 2006; 19–28.

[9] M. Kim, L. Bergman, T. Lau and D. Notkin. An ethnographic study of copy and paste programming practices in OOPL. Presented at ISESE '04: Proceedings of the 2004 International Symposium on Empirical Software Engineering.

[10] R. Koschke, Survey of research on software clones. Presented at In Proceedings of Dagstuhl Seminar 06301: Duplication, Redundancy, and Similarity in Software, 24pp., Dagstuhl.

[11] J. Krinke. A study of consistent and inconsistent changes to code clones. Presented at WCRE '07: Proceedings of the 14th Working Conference on Reverse Engineering.

[12] Rahman F, Bird C, Devanbu P. Clones: What is that smell? Proceedings of the 7th Working Conference on Mining Software Repositories, Cape Town, South Africa, 2010.

[13] Roy CK, Cordy JR, Koschke R. Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. Science of Computer Programming May 2009; 74(7):470–495.

[14] Kim M, Sazawal V, Notkin D, Murphy G. An empirical study of code clone genealogies. Foundations of Software Engineering, Proceedings of the the 10th European conference held jointly with 13th ACM SIGSOFT International symposium on Foundations of software engineering Sep 2005; :187–196.

[15] Thummalapenta S, Cerulo L, Aversano L, Di Penta M. An empirical study on the maintenance of source code clones. Empirical Software Engineering Mar 2009.

[16] S. Uchida, A. Monden, N. Ohsugi, T. Kamiya, K. Matsumoto, and H. Kudo. Software analysis by code clones in open source software. *The Journal of Computer Information Systems*, XLV(3):1–11, April 2005.