Clone Detection: How accurate is *your* data set?

Cory J. Kapser and Michael W. Godfrey
David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, Ontario, Canada
cjkapser, migod@uwaterloo.ca

Abstract

Duplication of code in software systems is considered to be a serious problem that can affect a systems maintainability and extendability. It is reported that 10-15% of code in a software system is involved in cloning. However, because of the difficultly of objectively measuring the number of false positives in a clone result set, the accuracy of these reports is difficult to evaluate. Although an important topic, little work has been done in the area of evaluating the accuracy of clone detection methods. In this paper we propose a study to estimate the number of false positives that are likely to be in a data set in an objective way by measuring the number of clones found in a large body of unrelated code. We also propose a method to measure the impact of external factors such as programing idioms and API protocols on the detected results set. The results of this work will provide tools and knowledge to better evaluate the current state of the art of clone detection research.

1 Introduction

Analysis and detection of clones in software has recently become a popular area of research. Code cloning, generally referred to as the practice of duplicating code within a software system, is considered a serious problem by many sources [4, 6, 8, 10, 15, 16, 18]. Some problems associated with code clones are unnecessary increase in code size, duplicated bugs and duplicated maintenance effort to fix them, introduction of unused code, and increased code complexity. If code cloning is not managed costs associated with maintaining and extending the system will increase needlessly.

Typically, clone detection tools report that 10-15% of the lines of code in a software system contribute to clones. In extreme cases, the duplication can be as high as 50% of the software system [8]. However, not all of these reported clones are related to the duplication of source code [12, 13]. In many cases false positives are also part of the result set. False positives are segments of code that are reported as clones but in fact are not. In many cases these matches are caused by segments of code with very simple and repetitive structure [13]. Many of these falsely reported clones can be removed from the result set using filters, but additional manual inspection of clones is required to refine the results further [13].

In addition to false positives, there is another type of clone not directly related to the duplication of source code that may be reported by clone detection tools. These clones, called "incidental clones", are segments of code that are similar in structure and function not because of explicit copy-and-paste activities but rather arise due to other factors such as programming idioms, API interactions, and the inherent structure of programs written in a programming language. These clones can be very difficult to filter, both manually and automatically, because their form and function may actually be related. For example, building a GUI is a highly repetitive task, and interactions with the API may result in many repeated calls to the same set of functions. In these cases, it is difficult to classify the cause of the clone as copy-and-paste or incidental.

It is important that we measure the proportion of clones in a result set that are false positives or incidental clones if

we wish to properly evaluate the effectiveness and accuracy of clone detection tools, yet little work has been done on this topic. This is largely because this type of evaluation would have required human subjects to classify the clones, a task that was found to be highly subjective [19]. With the recent existence of large source code repositories such as *csourcesearch.net* [1], we can now take an objective approach to measuring the amount of incidental cloning and false positives, there by giving us insight into the degree of true cloning within a software system.

This paper proposes an experiment that will measure the commonality amongst a very large set of unrelated open source projects taken from the *csourcesearch.net* project. Because these systems will be generally unrelated we expect that the code will be equally unrelated, giving us a baseline of false positives that are detected in unrelated code. This will provide insight about how many clones are detected amongst unrelated code when inspecting a software system, and giving us a way to more accurately estimate the amount of true cloning in a software system. In addition, we will also measure the effect of API protocols on clone detection results by measuring the amount of cloning that occurs between software that uses the same API or library. In this work we expect the commonalities amongst software systems to be low, providing further validation of the significance of cloning found within a software system.

2 Methodology

The goal of this study is to estimate the amount of false positives and incidental clones that exist in the results of a clone detection tool. We will do so by measuring the amount of clones that are detected amongst unrelated code, under the assumption that most clones that are detected will be false positives. This assumption was derived from the results of our previous work comparing source code of similar open source projects [2] where we found that the open source projects in our study did not share code, even though they were related in functionality.

The experiment will consist of two phases. In the first phase we will detect clones amongst a random sample of projects, giving us an estimate of false positives and incidental clones detected amongst unrelated code. In the second phase, our study subjects will consist of source code that is related to GUI construction. This phase will provide us with an estimate of the amount of incidental cloning that is detected by clone detection tools. For each phase, we will carry out the following steps:

1. Randomly select study subjects.
2. Detect clones between each study subject pair.
3. Detect clones within each study subject.
4. Measure overlap of clones within software systems with clones occurring between them.

Each of these steps will be discussed in more detail below.

2.1 Unrelated Code

For the purpose of this experiment, we will use 200 randomly selected projects, selected from the list of downloaded projects published by the author of *csourcesearch.net*. *csourcesearch.net* is an on-line searchable repository of a very large number of C projects. It allows the use to query the source code using a variety of mechanisms. To detect clones amongst files that include GUI libraries, we will use the "includes" search functionality. There will be no restriction on size of project. However, the source language will be restricted to C, the only language currently in the repository.

After selecting the projects, we will proceed to download the source and run clone detection tools on them. In this study we will use two clone detection techniques to gather our results, parametrized sting matching as described by Kamiya et al. [10] and exact match string matching as described by Ducasse et al. [8]. This will allow us to measure the impact of the detection technique on the results as well as provide us with a comparison of the amount and type of false positives that are detected by the two different approaches.

In our first step of clone detection we will only detect clones that occur across each possible pair of software systems. Because we expect most of the source code to be unrelated, most clones should in fact be false positives or incidental clones. From this set of results, we will record the average percent of commonality between each pair of systems and the average size of the clones. This will provide us with a baseline of the amount false positives that

occur in a results set from each of the clone detection techniques.

Through manual inspection of the results in this step, we will try to analyze the types of clones that tend to occur in both, in an effort to profile the types of code that cause false positives in the clone detection techniques we are using.

In our next steps we will detect the clones that occur within each project and measure the amount of code that occurs in both the set of clones across projects and within the projects. This will give us a further indicator as to how much code that is likely to be part of a false positive contributes to the detected clones in a software system.

2.2 GUI Code

In our next step, using *csourcesearch.net* we will search for any files in the repository that include header files from widget libraries such as GTK, GNOME Widgets, and xlib. Partitioning the files by project, we will detect the clones occurring across projects using the same libraries. By detecting the clones that occur between code using specialized libraries such as GUI libraries, we can gain insight into the degree of "incidental cloning" that is detected by clone detection tools. These clones will in many cases represent strategies or protocols required for the use of the libraries, something that can not be avoided. Perhaps the result of studying these clones can lead to further abstractions with the libraries themselves.

As in the first phase, will will detect clones within each of the projects as well. By measuring the overlap, well will gain insight into the contribution of "incidental clones" as part of a result set of detected clones.

3 Related Work

There is a wide variety of clone detection techniques that have been developed. These methods range from string comparison, metrics comparison, and program graph comparison strategies [4, 6, 8, 10, 16, 18, 9, 5, 17, 14]. Currently we propose to only use two of these methods of clone detection as a pilot study. The study we propose could be expanded to other clone detection techniques. Several case studies have been performed on cloning with a software system [3, 7, 10, 11, 12, 13], but none of these studies have considered measuring cloning across software systems.

There are very few studies that perform clone detection across software systems. Kamiya et al. [10] investigated the cloning across the source code of three different operating systems: Linux, FreeBSD, and NetBSD. Their analysis showed that there was about 20% cloning between FreeBSD and NetBSD, whereas there was less than 1% of the code cloned between Linux and FreeBSD or NetBSD. Because FreeBSD and NetBSD have the same origin, the cloning between them was not surprising. Because Linux was developed independently from the BSD systems, very little cloning was detected. In [2] we found similar results, finding that very few clones are detected across software that was not related. However, in both of these case, the study size is very small, making the results not generalizable. In addition, neither study considers the effects of using libraries such as GUI libraries on the clone detection results.

4 Conclusion

Previously it has been very difficult to objectively measure the amount of false positives are returned by a clone detection tool, yet this is important if we wish to confidently analyze the results of clone analysis and clone detection research. In this paper, we propose a study that will effectively find the lower limit of this value. In addition we also aim to measure the impact of the protocols required to use APIs on the clone detection results, helping us measure the effects of incidental cloning in a software system.

The results of this work will provide not only more insights into the accuracy of clone detection tools, but it also provides a platform from which we can investigate the weaknesses of tools, and also improve data filtering techniques. For example, from the resulting detected clones between software system and within software systems one may be able to train learning algorithms to classify true clones and false positives, something that we would like to research further.

References

[1] csourcesearch. "http://csourcesearch.net/", 2006.

[2] Raihan Al-Ekram, Cory Kapser, Richard Holt, and Michael Godfrey. Cloning by accident: An empirical study of source code cloning across software systems. In *2005 Intl. Symposium on Empirical Software Engineering (ISESE-05)*, 2005.

[3] G. Antoniol, U. Villano, E. Merlo, , and M. Di Penta. Analyzing cloning evolution in the linux kernel. In *Information and Software Technology 44(13)*, 2002.

[4] B. S. Baker. On finding duplication and near-duplication in large software systems. In *WCRE '95: Proceedings of the Second Working Conference on Reverse Engineering*, page 86, Washington, DC, USA, 1995. IEEE Computer Society.

[5] Magdalena Balazinska, Ettore Merlo, Michel Dagenais, Bruno Lague, and Kostas Kontogiannis. Advanced clone analysis to support object-oriented system refactoring. In *Proceedings of the 7th. Working Conference on Reverse Engineering*, pages 98–107, 2000.

[6] Ira D. Baxter, Andrew Yahin, Leonardo Moura, Marcelo Sant'Anna, and Lorraine Bier. Clone detection using abstract syntax trees. In *ICSM '98: Proceedings of the International Conference on Software Maintenance*, page 368, Washington, DC, USA, 1998. IEEE Computer Society.

[7] G. Casazza, G. Antoniol, U. Villano, E. Merlo, and M. Di Penta. Identifying clones in the linux kernel. In *First IEEE International Workshop on Source Code Analysis and Manipulation*, pages 92–100. IEEE Computer Society Press, 2001.

[8] Stéphane Ducasse, Matthias Rieger, and Serge Demeyer. A language independent approach for detecting duplicated code. In Hongji Yang and Lee White, editors, *Proceedings ICSM'99: International Conference on Software Maintenance*, pages 109–118. IEEE, 1999.

[9] J. H. Johnson. Substring matching for clone detection and change tracking. In *Proceedings of the International Conference on Software Maintanence*, pages 120–126, 1994.

[10] Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoue. Ccfinder: A multilinguistic token-based code clone detection system for large scale source code. In *Transactions on Software Engineering 8(7)*, pages 654–670. IEEE Computer Society Press, 2002.

[11] Cory Kapser and Michael W. Godfrey. Toward a taxonomy of clones in source code: A case study. In *Evolution of Large Scale Industrial Software Architectures*, 2003.

[12] Cory Kapser and Michael W. Godfrey. Aiding comprehension of cloning through categorization. In *Proc. of 2004 International Workshop on Principles of Software Evolution (IWPSE-04)*, pages 85–94, 2004.

[13] Cory Kapser and Michael W. Godfrey. Improved tool support for the investigation of duplication in software. In *The 2005 Intl. Conference on Software Maintenance*, 2005.

[14] Raghavan Komondoor and Susan Horwitz. Using slicing to identify duplication in source code. In *SAS '01: Proceedings of the 8th International Symposium on Static Analysis*, pages 40–56, London, UK, 2001. Springer-Verlag.

[15] K Kontogiannis. Evaluation experiments on the detection of programming patterns using software metrics. In *Proceedings of Working Conference on Reverse Engineering*, pages 44–55. IEEE Computer Society Press, 1997.

[16] Kostas Kontogiannis, Renato DeMori, Ettore Merlo, M. Galler, and M. Bernstein. Pattern matching for clone and concept detection. *Autom. Softw. Eng.*, 3(1/2):77–108, 1996.

[17] Jens Krinke. Identifying similar code with program dependence graphs. In *Proc. Eigth Working Conference on Reverse Engineering*, pages 301–309, 2001.

[18] J. Mayrand, C. Leblanc, and E. Merlo. Experiment on the automatic detection of function clones in a software system using metrics. In *Proceedings of the International Conference on Software Maintenance*, pages 244–253. IEEE Computer Society Press, 1996.

[19] Andrew Walenstein, Nitin Jyoti, Arun Lakhotia, Junwei Li, and Yun Yang. Human judgment of function clones: Problems for automated tool evaluation. In *10th Working Conference on Reverse Engineering (WCRE'03)*, 2005.