Studying Code Clones Through Concolic Analysis

Daniel E. Krutz
Department of Software Engineering

October 3, 2013

1 Motivation

Code clones are extremely widespread in software development. It has been estimated that clones typically comprise between 5-30% of an application's source code [1] [15] [7]. Code clones may adversely affect the software development process for several reasons. They often raise the maintenance costs of a software project since alterations may need to be done several times [5]. Additionally, unintentionally making inconsistent bug fixes to cloned code across a software system will likely cause further system faults [2].

There are four types of code clones. Type-1 clones are the simplest and represent identical code except for differences in whitespace, comments and layout. Type-2 clones are syntactically similar except for variations in identifiers and types. Type-3 clones are two segments which differ due to altered or removed statements. Type-4 clones are the most difficult to detect and represent two code segments which significantly differ syntactically, but produce identical results when executed [4].

Although there have been a substantial number of proposed clone detection techniques [12], to our knowledge only two known tools are able to reliably discover type-4 clones. These are Memory Comparision-based Clone Detector (MeCC) [6] and our tool, Concolic Code Clone Detection (CCCD) [10]. There are several reasons why most existing techniques have struggled at finding type-4 clones. Some of which include an over-reliance on the semantic or syntactic properties of the source code. The ability to detect type-4 clones is critical since these clones not only comprise a significant portion of the functional redundancy of an application, but may be the most problematic as well [11] [17].

Assessing the quality of any proposed clone detection technique is imperative for demonstrating its value. These evaluations typically involve two primary criteria. The first is the efficiency of the proposed method, both in terms of execution time and required system resources. The more difficult criteria to evaluate is the quality of the detection technique. Most commonly, precision and recall are measured against an oracle of predefined clones. Constructing such an oracle is difficult. While manually verifying simpler, type-1 clones is a time-consuming, but relatively easy task, manually identifying the more complicated types of clones becomes increasingly difficult to the point of being practically impossible. Recent research has shown that oracles created using a solely manual process are typically very inaccurate [16]. The only known clone oracles which contain all four types of clones and have been verified both manually and by clone detection tools tools were created by Krawitz [8] and Roy *et al.* [11]. However, these two oracles are only comprised of a combined 24 methods and are unsuitable for a large scale analysis.

There has been a significant amount of research on how code clones affect the software development process [5] [3] [14]. However, none of this work considers type-4 clones in their analysis. Only recently have there been any tools which have been capable of discovering these types of clones. In addition to software development, clone detection tools have been applied to other areas of computing such as malware detection and analysis [13].

The first goal of this proposal is to evaluate our clone detection tool, CCCD, against other leading techniques. An empirical analysis will then be performed on several large open source applications to examine how type-3 and type-4 clones affect the software development process. Finally, a large code clone oracle which contains all four types of clones and has been rigorously verified by using both existing tools and by manual analysis will be created and made publicly available. This will not only assist our research, but will assist future researchers as well.

2 Execution Plan and Outcomes

There are three primary components to the proposed project.
1) Analyze the effectiveness of concolic analysis for clone detection against numerous existing tools.
2) Perform an empirical analysis to discover how type-3 and type-4 clones affect the software engineering life cycle.
3) Build a robust code clone oracle which is publicly available.

2.1 Effectiveness of Concolic Analysis

Concolic analysis will first be analyzed to determine its effectiveness in discovering clone clones. Since we have already demonstrated that concolic analysis is capable of discovering clones on a small scale [10] [9], we will next evaluate concolic analysis using a more robust set of test data. Concolic analysis will be run against several open source applications to determine what types of clones it is capable of finding on a much larger scale. Some of these candidates include Apache, Python and PostgreSQL. Next, several leading clone detection tools will be run against this same code base. Some of these other tools include CloneDR[1], Simian[2], and MeCC[3]. The evaluation metrics will include analysis time, precision and recall, and types of clones discovered.

2.2 Empirical Analysis

An empirical analysis will be conducted on the version control systems of several open source applications to determine the effect all types of code clones have on the the software development process. This will be accomplished by first running CCCD on the source code of these applications to identify code clones. Next, a qualitative and quantitative analysis will be performed on the source code to answer some questions about how code clones affect software development. Some of these questions include why the clones were created, if more bugs exist in these clones, do bugs occur consistently across the clones, do vulnerabilities exist across clones, and if vulnerabilities are linked to clones. A tool will likely need to be created to assist with this process. While there has been work in determining how type-1 and type-2 clones affect the software development process [5] [**?**], to our knowledge no work has been conducted on how more complicated type-3 and type-4 clones affect software development.

2.3 Clone Oracle Creation

A code clone oracle containing all four types will then be generated and will be made publicly available for other researchers. While several other clone oracles exist, this will be the first significantly sized oracle to contain all four types of clones. We will create a code clone oracle containing all four types of clones using several verification techniques to ensure maximum quality. We will use much of the data used in our comparison of clone detection tools to assist in creating this oracle. First we will manually analyze the selected open source applications for clones. In order to do this in an efficient, but effective manner we will use a partially developed tool we have created known as GraphicDiff. This tool will automatically load each method from the targeted codebase and display them side by side for analysis. A simple interface will allow the user to record if the two displayed methods represent a clone, and if so what kind. They may then cycle onto the next set of methods. All methods in the target application will be displayed to the user in this manner. In order to ensure the highest level of quality using this manual technique, several researchers familiar with code clones will independently perform this analysis and discuss any discrepancies.

The next step will involve analyzing this same source code using the existing clone detection tools from the comparison process. For clones identified using these tools, but not by manual analysis, the source code will be again manually examined to ensure that no clones were missed during the initial manual analysis step. A database of all clones identified by these tools will be created and compared with the clones as identified by the manual process. Discrepancies will be discussed and analyzed by several researchers. Based upon the ability of each tool in detecting clones, all identified clones will likely need to be manually analyzed and verified. Once the oracle has been created, the results will be posted online in an easily usable format and will available for future work by this project, and to other researchers.

3 Relevant Experience

The proposed project is closely related to my research expertise and my PhD thesis. For my dissertation [9] I proposed and demonstrated the effectiveness of concolic analysis for clone detection on a fairly small scale. In a recent publication [10], we used this technique to develop a tool for discovering code clones, Concolic Code Clone Detection (CCCD) [10]. This research demonstrated the ability of CCCD to effectively discover type-4 code clones, something that only one other tool (MeCC) is able to do. Based upon my dissertation work, our tool development and subsequent research, I believe that we have the necessary qualifications to successfully carry out the proposed project.

[1] http://www.semdesigns.com/products/clone/
[2] http://www.harukizaemon.com/simian/
[3] http://ropas.snu.ac.kr/mecc/

4 Proposal Goals

The findings of this project are anticipated to produce innovative and highly visible results and are anticipated to produce publications in venues such as FSE, ICSE, MSR or FASE. Several companies have expressed interest in such a robust concolic analysis process for clone, and possibly malware detection. Additionally, they are enthusiastic about a technique which would be able to analyze the amount of similarity between the source code of multiple applications to determine if functionality has been illegally replicated or the amount of new functionality a company would be achieving with the acquisition of a another software organization or package. The goal of this project lay a foundation for future research and demonstrate results to support a larger project which will be proposed to the NSF and industry for funding.

5 Budget

[Dan says: How do these number look?- Look at what I am requesting here] [Dan says: Summer?] [Dan says: Can I get a worker sooner?]

I am requesting a total of $6,750 for this project. The project will be completed with the assistance of an undergraduate or graduate student over the course of both the Spring and Summer semesters. The requested budget is based on having the student work 10 hours per week for 15 weeks @ $15/hr (totaling $2,250) during the Spring term and 20 hours per week for 15 weeks during the Summer semester @$15/hr (totaling $4,500).

The student will assist in comparing concolic analysis against leading clone detection tools and by performing an empirical analysis on the discovered type-3 and type-4 clones to further analyze how they affect the software engineering process. Finally, they will assist in creating the software needed to generate the clone oracle, and building the web application to share this oracle information.

References

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

[2] Florian Deissenboeck, Benjamin Hummel, and Elmar Juergens. Code clone detection in practice. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2*, ICSE '10, pages 499–500, New York, NY, USA, 2010. ACM.

[3] Nils Gode. Evolution of type-1 clones. In *Proceedings of the 2009 Ninth IEEE International Working Conference on Source Code Analysis and Manipulation*, SCAM '09, pages 77–86, Washington, DC, USA, 2009. IEEE Computer Society.

[4] Nicolas Gold, Jens Krinke, Mark Harman, and David Binkley. Issues in clone classification for dataflow languages. In *Proceedings of the 4th International Workshop on Software Clones*, IWSC '10, pages 83–84, New York, NY, USA, 2010. ACM.

[5] Elmar Juergens, Florian Deissenboeck, Benjamin Hummel, and Stefan Wagner. Do code clones matter? In *Proceedings of the 31st International Conference on Software Engineering*, ICSE '09, pages 485–495, Washington, DC, USA, 2009. IEEE Computer Society.

[6] Heejung Kim, Yungbum Jung, Sunghun Kim, and Kwankeun Yi. Mecc: memory comparison-based clone detector. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 301–310, New York, NY, USA, 2011. ACM.

[7] Miryung Kim, Vibha Sazawal, David Notkin, and Gail Murphy. An empirical study of code clone genealogies. *SIGSOFT Softw. Eng. Notes*, 30(5):187–196, September 2005.

[8] Ronald M. Krawitz. *Code Clone Discovery Based on Functional Behavior*. PhD thesis, Nova Southeastern University, 2012.

[9] Daniel E. Krutz. *Concolic Code Clone Detection*. PhD thesis, Nova Southeastern University, 2012.

[10] Daniel E. Krutz and Emad Shihab. Cccd: Concolic code clone detection. In *Reverse Engineering (WCRE), 2013 20th Working Conference on*, 2013.

[11] Chanchal K. Roy, James R. Cordy, and Rainer Koschke. Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. *Sci. Comput. Program.*, 74(7):470–495, May 2009.

[12] Chanchal Kumar Roy and James R. Cordy. A survey on software clone detection research. *SCHOOL OF COMPUTING TR 2007-541, QUEENS UNIVERSITY*, 115, 2007.

[13] Andreas Saebjornsen, Jeremiah Willcock, Thomas Panas, Daniel Quinlan, and Zhendong Su. Detecting code clones in binary executables. In *Proceedings of the eighteenth international symposium on Software testing and analysis*, ISSTA '09, pages 117–128, New York, NY, USA, 2009. ACM.

[14] Ripon K. Saha, Chanchal K. Roy, Kevin A. Schneider, and Dewayne E. Perry. Understanding the evolution of type-3 clones: an exploratory study. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 139–148, Piscataway, NJ, USA, 2013. IEEE Press.

[15] Sandro Schulze, Sven Apel, and Christian Kästner. Code clones in feature-oriented software product lines. *SIGPLAN Not.*, 46(2):103–112, October 2010.

[16] Andrew Walenstein, Nitin Jyoti, Junwei Li, Yun Yang, and Arun Lakhotia. Problems creating task-relevant clone detection reference data. In *Proceedings of the 10th Working Conference on Reverse Engineering*, WCRE '03, pages 285–, Washington, DC, USA, 2003. IEEE Computer Society.

[17] Yang Yuan and Yao Guo. Cmcd: Count matrix based code clone detection. In *Proceedings of the 2011 18th Asia-Pacific Software Engineering Conference*, APSEC '11, pages 250–257, Washington, DC, USA, 2011. IEEE Computer Society.