Semantic clone detection and benchmarking

Farouq Al-omari

July 13, 2018

Permission to Use

In presenting this thesis in partial fulfilment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Computer Science
176 Thorvaldson Building
110 Science Place
University of Saskatchewan
Saskatoon, Saskatchewan
Canada
S7N 5C9

Abstract

Developers copy and paste their code to speed up the development process. Some-times, they copy code from an open source system and reuse it with or without modifications. The resulted similar or identical code fragments in terms of syntax or semantic are called code clones. Software cloning researches indicate the existence of code clones in all software systems; on average 5% to 20% of software code is cloned. Due to clones impact, either positive or negative impacts, its important to locate, track and manage them in the source code. Many techniques for detecting code clones have been proposed. However, a recent study shows that existing techniques have a limitation in detecting semantic clones. In this research, we proposed different techniques using intermediate language and ontology to detect clones across programming languages and semantic clones. We evaluated the efficiency of our techniques and compare it to start-of-art detection tools. Finally, we proposed a methodology to create a functional clone oracle that help researcher in this area to evaluate their tools.

Acknowledgements

Acknowledgements go here. Typically you would at least thank your supervisor.

This is the thesis dedication (optional)

Contents

List of Tables

List of Figures

List of Abbreviations

SCUBA	Self Contained Underwater Breathing Apparatus
LOF	List of Figures
LOT	List of Tables

Chapter 1

Introduction

1.1 Background

Software clones are defined as similar (near-miss) or identical (identical clones) code fragments in terms of syntax or semantic. Usually, these code fragments result from the practice of programmers copying and pasting which produces identical clones. However, if the copied code fragments have minor modifications, they result in near- miss clones. The case of having major modifications to these clones will result in their disappearance. Conversely, some clones are unintentionally introduced into software systems due to the programmer practice to achieve common tasks or due to the use of library or API to implement common tasks. When code fragments have the same functionality, regardless of their syntactic, they called semantic clones.

Software code cloning offers benefits during the development process. Usually, developers reuse their own code to save the time of rewriting, or they reuse others code to overcome some programming and design limitations[35]. In some cases, the cloned code might have a serious problem; i.e. bugs that need more testing or updates in the maintenance phase [11, 32]. On the other hand, skilled developers pay more attention in order to choose higher quality, well tested, and bug free code to clone [4, 23]. Practitioners have two different opinions about whether clones are harmful [2, 7, 5, 19, 28, 30] or not [21, 4, 5, 20, 12]. As a result, some studies target software clone harmfulness/usefulness [15]. For example, [23, 33, 29] compared the co-changes of cloned to non-cloned code. Other studies compared the stability of cloned and non-cloned code [16, 13, 31, 25].

Over the decades, practitioners have proposed different techniques to detect both syntactic and semantic clones. Detecting syntactic clone is easier than semantic clones. In syntactic clone, the source code is normalized then transformed into other representations (token, tree, or vectors) before it used for comparison. However, in semantic clone detection more normalization needs to be done. For instance, dependencies and relationships (PDG) have to be identified and represented, and the functionality should be captured and used in comparing code units. A recent study shows that existing techniques and tools have some limitation in the detection of semantic clones (functional clones) [39].

More recently there has been an ongoing trend towards multi-language software development to take advantage of different programming languages [24]; specifically in the .NET context. For multi-language development, two key usage scenarios can be distinguished: (1) combining different programming languages

within a single, often large and complex system, and (2) the use of several languages for re-implementation of a current system to support new client, application, or due to non-technical reasons. As a result, the ability to detect and manage similar code reuse patterns that might exist in these multiple languages systems becomes essential. While many clone detection tools are capable of supporting different programming languages, they lack actual cross-language support during detection time. Consequently, these tools only detect clones in one program language at the time, and do not detect clones that span over multiple programming languages.

The accuracy of both emerging and proposed techniques are needed to be evaluated in detecting all types of clones. Three major techniques are used for evaluation: (1) manual inspection of reported clones to identify true positives and false positives, (2) injecting the source code with artificially generated clones to measure how many clones the tool(s) are able to detect [37, 34], and 3) using benchmarks, for already identified and known clones in the system [26, 3].

The rest of the proposal is organized as follows: Background topics of my research that includes clone definition, intermediate language, Ontology, ontology matching, and matching algorithms which are presented in section 2. Section 3 presents the related work to our research. Section 4 states the thesis statement. Section 5 presents our study in clone detection across programming languages. Section 6 describes a proposed study in detecting semantic clones. Another proposed technique in the detection of semantic clones is presented in section 7. In section 8, we describe a technique to build a clone oracle. Finally, Section 9 concludes the paper.

Chapter 2

Background

2.1 Introduction

2.1.1 Semantic clone definition

A code fragment in the source code that is identical, or similar, to another code fragment in the code base is considered code clone to the second and both called clone pair. This definition is based on concept of similarity. Within existing literature, the following categorization of clone definition have been widely acceptable:

Type I: Identical code fragments except for variations in

whitespace (may also have variations in layout) and comments.

Type II: Structurally/syntactically identical fragments except for

variations in identifiers, literals, types, layout and comments.

Type III: Copied fragments with further modifications.

Statements can be changed, added or removed in addition to

variations in identifiers, literals, types, layout and comments.

Type IV: Two or more code fragments that

perform the same computation but are implemented through different

syntactic variants.

Type I, Type II, and Type III clones are based on textual similarity. A code's fragments are considers clones if they are textually similar, even if they are functionally different. Textual clones are more common in software code base because they are usually the result of copy/paste practice. Conversely, functional similar code fragments are considered clones even they are textually (syntactically) different. Semantic clones are difficult to detect since they could be implemented by different syntactics and any single change in a code fragment could change the meaning (functionality) of the fragment.

Type IV clones (some times referred to as functional clones or dependence clones) are the results of semantic similarity between two or more code fragments. In this type of clones, the cloned fragment is not sharing the syntax of structure from the original. Two code fragments could be developed by two different programmers to do the same kind of task, making the code fragments similar in their functionality. The best example for semantic clones are sorting algorithms (merge sort, quick sort, and bubble sort). All of them do the same functionality and are implemented in a different ways, syntactic, and structure. To simplify, the following two functions do the same logic (swapping two variables). However, implemented in two different syntactic.

```
public static int[] swap(int[] a, int i, int j)
{
int x = a[i];
a[i] = a[j];
a[j] = x;
return a;
}

public static int[] swapNoTemp(int[] a, int i, int j)
{
a[i] = a[i]+a[j];
a[j] = a[i]-a[j];
a[i] = a[i]-a[j];
return a;
}
```

Some practitioners consider syntactic clones that have similar functionality to be semantic clones while others define semantic clones as functionally similar and syntactically different. [35] defines semantic clones as a functionally similar clones and implemented using a different syntactics; while [6, 8, 9] consider functionally similar code fragments as a semantic clones regardless of its syntactic.

They key challenge is to detect (semantic) clones that are not detectable by the most mature syntactic detectors such as: NICAD [36], CCFinder [19], and CloneDR [1]. A numbers of techniques are proposed in the literature to detect such clones [14, 22, 9, 10] but most of these are not based on a solid definition of semantic clones.

2.1.2 Common Intermediate Language

The Common Intermediate Language (CIL), also known as Microsoft Intermediate Language (MSIL), is a stack-based virtual machine. Unlike other VMs, MSIL is designed to support a wide range of languages (C#, Visual basic, Visual C++) and platforms (Visual Studio and mono). When compiling the source code,

the compiler translate the source code into MSIL, a machine independent set of instructions. Then MSIL converted into CPU-specific code using just-in-time compiler. In the following proposal, we use Intermediate Language IL, disassembled code, bytecode, and binary code to refer to CIL.

CIL is a low level language that includes a set of 17 data types. It also includes a total of 255 instructions that are classified into loading, storing, initializing, calling methods on objects, arithmetic and logical operations, control flow, direct memory access, exception handling, and other operations. These instructions are considered a rich source of knowledge of source code structure as well as semantic . For example, the instructions bgt, bgt.un, ble, ble.un, blt and blt.un are all branching instructions. Therefore, the relationship (semantically) between these instructions could be easily measured by their textual similarity.

2.1.3 Ontology

Ontology is a conceptual model to represent knowledge of an application domain by first defining the relevant concepts of the domain and then using these concepts to specify properties of objects and individuals occurring in the domain (Baader, 2003).

In this work we used the disassembled code and the source code itself to build a knowledge model of the code base. We extracted entities, facts, relations, and other components from disassembled code and built a simple formal ontology. Methods are the main entities of the constructed ontology since we are targeting clones at method level.

Ontologies could be defined in different ontology languages, the most popular one is OWL. Defined ontologies usually consist of the following entities:

Classes or Concepts are the main entities of an ontology. For example, it could be book, course or student. In code base domain, it could be a file, class, method or variable.

Individuals are instances of classes in the domain. All files in the source code are individuals.

Relations identify the relationships between individuals.

Data types specify values such as string and integer.

Data values are simple values such as file name and method name.

Specialisation represents inclusion relationship between two classes.

Exclusion represents emptiness of intersection between classes.

Instantiation represents membership between classes and individuals or values and data types.

2.1.4 Ontoloy Matching

Ontology matching is the process of finding the relationships or correspondences between the entities of two different ontologies. In Figure **??** this correspondence represented in a dashed arrow. Sometimes practitioners

refer to it as ontology alignment or mapping. Ontologies represented in a hierarchy structure, XML alike, and it used to basically describe knowledge in semantic web, web services, or knowledge in other domains. Ontology mapping plays a major role in different applications such as, information sharing, query answering, data integration, etc...

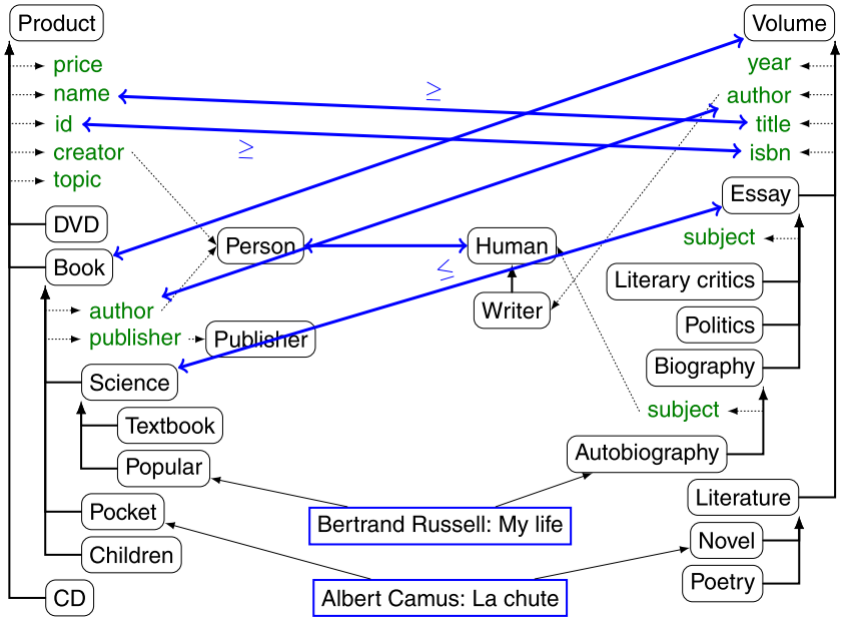

Figure 2.1

Ontology matching techniques are based on finding the correspondences between ontologies' entities, structure, or relations. The amount of published research in this area is remarkable. Ontology matching techniques are classified into: string-based, structure-based, constraint-based, instance-based, and model-based techniques. For the purpose of our research, any of state-of-art matching tools could be used. However, we implement the matching technique that we used based on the type of ontology designed and its semantics.

2.1.5 Matching Algorithms

In this section we demonstrate string matching algorithms used in our alignment technique. There are many techniques that could be used to compare strings depending on the way information is represented in the strings. For example, a set of letters, a sequence of letters, an erroneous sequence of letters, a set of words, or a sequence of words. In this work, we used different string similarity measurements. We used Levienshtien distance for sequence of letter strings, longest common subsequence for a sequence of words, Jaccard similarity for a set of words, and SimHash which is based on fingerprint similarity. Other similarity measures could be use for the purpose of matching such as Hamming distance, n-gram similarity, Euclidean, cosine, JaroâĂŞWinkler, MongeâĂŞElkan, TFIDF and soundex.

Levenshtein Distance (levDist), also called Edit Distance, is defined as the minimum number of insertions,

deletions, and substitutions of characters required to transform one string into the other [27]. Levenshtein Distance provides the minimum total cost of operations (Op) between two string as a single number value. We use LevSim (Eq. 1) as a similarity measurement between two strings. this is how to add a citation [17]

$$LevSim(s1, s2) = 1 - \frac{LevDist(s1, s2)}{max(size(s1), size(s2))} \quad (1)$$

The Longest Common Subsequence (LCS) [17] algorithm detects the longest common subsequence between two strings. For example, consider the following two sequences of characters.

S1 =AABBBCDABCDDAABD

S2 = DDABCCCDAABBBDAC

For the above example, the LCS among the S1 and S2 sequence is ABCDABDA. For our technique we consider each word as one unit and we used LCS to find the longest common tokens and measure the similarity between two token blocks (Eq. 2).

$$LCS\_Sim(s1, s2) = \frac{LCS(s1, s2)}{\frac{size(s1) + size(s2)}{2}} \quad (2)$$

Jaccard similarity (Jacc), also referred to as gloss overlap between two strings, is defined as the intersection of their character set divided by the union of their character set [18] (Eq. 3). We used Jaccard coefficient to measure the similarity (overlap) between two sets of token.

$$Jacc(s1, s2) = \frac{|s1 \cap s2|}{|s1 \cup s2|} \quad (3)$$

SimHash-based Clone Detection SimHash algorithm constitutes the core of SimCad [38]. It generates a 64-bit fingerprint, which we use to detect clones based on their fingerprint similarities. The algorithm uses Charikar's [18] hash function where the Hamming Distance is used as the crucial configuration parameter. The Hamming Distance represents the number of positions at which the corresponding bits are different between two fingerprints. A Hamming distance of zero corresponds to identical fingerprints and therefore also to Type-1 clones, while a Hamming distance larger than zero reflects near-miss clones.

References

[1] I.D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier. Clone detection using abstract syntax trees. In *Proceedings. International Conference on Software Maintenance (Cat. No. 98CB36272)*, pages 368–377. IEEE Comput. Soc, 1998.

[2] Saman Bazrafshan. No clones, no trouble? In *2013 7th International Workshop on Software Clones (IWSC)*, pages 37–38. IEEE, may 2013.

[3] Stefan Bellon, Rainer Koschke, Giulio Antoniol, Jens Krinke, and Ettore Merlo. Comparison and Evaluation of Clone Detection Tools. *IEEE Transactions on Software Engineering*, 33(9):577–591, sep 2007.

[4] J. R. Cordy. Comprehending Reality: Practical Challenges to Software Maintenance Automation. jan 2003.

[5] Michael W. Godfrey Cory J. Kapser. Supporting the Analysis of Clones in Software Systems: A Case Study. *JOURNAL OF SOFTWARE MAINTENANCE AND EVOLUTION: RESEARCH AND PRACTICE*, 18:2006, 2006.

[6] N. Davey, P. Barson, S. Field, R. Frank, and D. Tansley. The development of a software clone detector. 1995.

[7] S. Ducasse, M. Rieger, and S. Demeyer. A language independent approach for detecting duplicated code. In *Proceedings IEEE International Conference on Software Maintenance - 1999 (ICSM'99). 'Software Maintenance for Business Change' (Cat. No.99CB36360)*, pages 109–118. IEEE, 1999.

[8] R Elva. Detecting Semantic Method Clones in Java Code Using Method IOE-Behavior. 2013.

[9] R Elva and GT Leavens. Semantic clone detection using method IOE-behavior. *Proceedings of the 6th International Workshop on*, 2012.

[10] M Gabel, L Jiang, and Z Su. Scalable detection of semantic clones. *2008 ACM/IEEE 30th International*, 2008.

[11] Reto Geiger, Beat Fluri, Harald C. Gall, and Martin Pinzger. *Fundamental Approaches to Software Engineering*, volume 3922 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, Berlin, Heidelberg, mar 2006.

[12] Nils Gode and Jan Harder. Clone Stability. In *2011 15th European Conference on Software Maintenance and Reengineering*, pages 65–74. IEEE, mar 2011.

[13] Jan Harder and Nils Göde. Cloned code: stable code. *Journal of Software: Evolution and Process*, 25(10):1063–1088, oct 2013.

[14] Yoshiki Higo and Shinji Kusumoto. Code Clone Detection on Specialized PDGs with Heuristics. In *2011 15th European Conference on Software Maintenance and Reengineering*, pages 75–84, Higo2011, mar 2011. IEEE.

[15] Wiebe Hordijk, María Laura Ponisio, and Roel Wieringa. Harmfulness of code duplication: a structured review of the evidence. pages 88–97, apr 2009.

[16] Keisuke Hotta, Yui Sasaki, Yukiko Sano, Yoshiki Higo, and Shinji Kusumoto. An Empirical Study on the Impact of Duplicate Code. *Advances in Software Engineering*, 2012:1–22, jan 2012.

[17] James W. Hunt and Thomas G. Szymanski. A fast algorithm for computing longest common subsequences. *Communications of the ACM*, 20(5):350–353, may 1977.

[18] P Jaccard. Distribution de la Flore Alpine: dans le Bassin des dranses et dans quelques régions voisines. 1901.

[19] T. Kamiya, S. Kusumoto, and K. Inoue. CCFinder: a multilinguistic token-based code clone detection system for large scale source code. *IEEE Transactions on Software Engineering*, 28(7):654–670, jul 2002.

[20] Cory Kapser and Michael W. Godfrey. Aiding Comprehension of Cloning Through Categorization. pages 85–94, sep 2004.

[21] Cory J. Kapser and Michael W. Godfrey. âĂIJCloning considered harmfulâĂİ considered harmful: patterns of cloning in software. *Empirical Software Engineering*, 13(6):645–692, jul 2008.

[22] Iman Keivanloo, Chanchal K. Roy, and Juergen Rilling. SeByte: A semantic clone detection tool for intermediate languages. In *2012 20th IEEE International Conference on Program Comprehension (ICPC)*, pages 247–249. IEEE, jun 2012.

[23] Miryung Kim, Vibha Sazawal, and David Notkin. An empirical study of code clone genealogies. *ACM SIGSOFT Software Engineering Notes*, 30(5):187, sep 2005.

[24] Kostas Kontogiannis, Panos Linos, and Kenny Wong. Comprehension and Maintenance of Large-Scale Multi-Language Software Applications. In *2006 22nd IEEE International Conference on Software Maintenance*, pages 497–500. IEEE, sep 2006.

[25] Jens Krinke. Is Cloned Code More Stable than Non-cloned Code? In *2008 Eighth IEEE International Working Conference on Source Code Analysis and Manipulation*, pages 57–66. IEEE, sep 2008.

[26] Daniel E. Krutz and Wei Le. A code clone oracle. In *Proceedings of the 11th Working Conference on Mining Software Repositories - MSR 2014*, pages 388–391, New York, New York, USA, 2014. ACM Press.

[27] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.

[28] Simone Livieri, Yoshiki Higo, Makoto Matsushita, and Katsuro Inoue. Analysis of the Linux Kernel Evolution Using Code Clone Coverage. In *Fourth International Workshop on Mining Software Repositories (MSR'07:ICSE Workshops 2007)*, pages 22–22. IEEE, may 2007.

[29] Angela Lozano, Michel Wermelinger, and Bashar Nuseibeh. Evaluating the Harmfulness of Cloning: A Change Based Experiment. In *Fourth International Workshop on Mining Software Repositories (MSR'07:ICSE Workshops 2007)*, pages 18–18. IEEE, may 2007.

[30] Manishankar Mondal, Chanchal K. Roy, Md. Saidur Rahman, Ripon K. Saha, Jens Krinke, and Kevin A. Schneider. Comparative stability of cloned and non-cloned code. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing - SAC '12*, page 1227, New York, New York, USA, mar 2012. ACM Press.

[31] Manishankar Mondal, Chanchal K. Roy, and Kevin A. Schneider. An empirical study on clone stability. *ACM SIGAPP Applied Computing Review*, 12(3):20–36, sep 2012.

[32] Damith C Rajapakse and Stan Jarzabek. An investigation of cloning in web applications. In *Special interest tracks and posters of the 14th international conference on World Wide Web*, WWW '05, pages 924–925, New York, NY, USA, 2005. ACM.

[33] Harald C. Gall Martin Pinzger Reto Geiger, Beat Fluri. Relation of code clones and change couplings.

[34] Chanchal K. Roy and James R. Cordy. A Mutation/Injection-Based Automatic Framework for Evaluating Code Clone Detection Tools. In *2009 International Conference on Software Testing, Verification, and Validation Workshops*, pages 157–166. IEEE, 2009.

[35] Chanchal Kumar Roy and James R Cordy. A Survey on Software Clone Detection Research. *Computing*, 541:115, 2007.

[36] C.K. Roy and J.R. Cordy. NICAD: Accurate Detection of Near-Miss Intentional Clones Using Flexible Pretty-Printing and Code Normalization. In *2008 16th IEEE International Conference on Program Comprehension*, pages 172–181. IEEE, jun 2008.

[37] Jeffrey Svajlenko, Judith F. Islam, Iman Keivanloo, Chanchal K. Roy, and Mohammad Mamun Mia. Towards a Big Data Curated Benchmark of Inter-project Code Clones. In *2014 IEEE International Conference on Software Maintenance and Evolution*, pages 476–480. IEEE, sep 2014.

[38] Md. Sharif Uddin, Chanchal K. Roy, Kevin A. Schneider, and Abram Hindle. On the Effectiveness of Simhash for Detecting Near-Miss Clones in Large Scale Software Systems. In *2011 18th Working Conference on Reverse Engineering*, pages 13–22. IEEE, oct 2011.

[39] Stefan Wagner, Asim Abdulkhaleq, Ivan Bogicevic, Jan-Peter Ostberg, and Jasmin Ramadani. How are functionally similar code clones syntactically different? An empirical study and a benchmark. *PeerJ Computer Science*, 2:e49, mar 2016.

Appendix A

Sample Appendix

Stuff for this appendix goes here.

Appendix B

Another Sample Appendix

Stuff for this appendix goes here.