Dolly or Shaun? A Survey to Verify Code Clones Detected using Similar Sequences of Method Calls

Alexandre Paiva, Johnatan Oliveira, Eduardo Figueiredo

Department of Computer Science – Federal University of Minas Gerais (UFMG)
Belo Horizonte, Brazil

ampaiva@gmail.com, {johnatan-si, figueiredo }@dcc.ufmg.br

Abstract. *Software developers usually copy and paste code from one part of the system to another. This practice, called code clone, spreads the same logic over the system, hindering maintenance and evolution tasks. Several methods have been proposed to detect code clones. This paper presents a survey to evaluate code clones automatically detected by analyzing similar sequences of method calls. The survey was conducted with 25 developers that, by means of code inspection, analyzed the code clone candidates. Results showed that more than 90% of participants confirmed the code clones. Therefore, we conclude that sequences of method calls are good indicators of code clone.*

1. Introduction

A common decision made by a software developer when coding is to reuse existing code as a reference [Rattan et al., 2013]. Reasons vary and are not mutual exclusive. They include getting an idea of how to solve a specific problem or using something already tested [Ducasse et al., 1999]. Copying existing code fragments and pasting them with or without modifications into other parts of the system is called code clone, an important area of software engineering research [Rattan et al., 2013; Ducasse et al., 1999; Roy et al., 2009]. This practice is considered a bad smell [Fowler, 1999] since it leads to problems during software development and maintenance tasks. The reason is that code clones duplicate logic and, therefore, increase points of refactoring and error fixing.

In order to detect code clones, several methods and tools have been proposed. The most common strategies are based on tree [Wahler et al., 2004], program dependency graph (PDG) [Krinke, 2001; Komondoor and Horwitz, 2001], text [Johnson, 1993], token [Hummel et al., 2010], metrics [Marinescu, 2004; Kontogiannis, 1997], and hybrid techniques [Göde and Koschke, 2011]. For instance, Baxter et al. (1998) propose a tree-based clone detection technique. In their approach, a program is first transformed on an abstract syntax tree (AST) and, for every sub-tree in the AST, similar sub-trees are compared pairwise.

In a previous work [Paiva, 2016], Paiva observed that code clones can also be detected by comparing similar sequences of method calls in different parts of the system. That is, when a code is copied from one part of the system and pasted into another, all instructions of the original code come together. Therefore, a code clone is a repetition of sequence of instructions in different points of the system. After the code clone operation, the parts can evolve independently, receiving different changes. These changes make harder to detect the code clone. However, part of the original sequence of method calls is supposed to be preserved, since the original computation is one of the reasons for code cloning.

Although several different methods for code clone detection have been proposed, we lack empirical knowledge about the developers' subjective view of detected code clone. In other words, we need to know whether developers agree with code clone instances detected by these methods. To fill this gap, this paper presents and discusses the results of a survey with 25 software developers (Section 3). The survey contains 12 random samples of code clones automatically detected using similar sequences of method calls in 5 software systems. After filling a characterization form with their background profile, participants were asked to answer whether each sample (i.e., pair of code) is clone or not. In addition, the subjects should explain their reasons of choice.

The preliminary results (Section 4) show that, in general, more than 90% of subjects agree with the detected code clones. Subjects did not confirm only 2 out of 12 code clone candidates. Even these cases, the decision whether they are code clones or not was not a consensus. Indeed, in these two cases, subjects were divided half to half in their opinions. Therefore, results so far indicate that sequences of method calls is a prominent strategy for detecting code clones. This paper also discusses limitations of the study (Section 5) and related work (Section 6). Section 7 concludes this paper and points out directions for future work.

2. Background

This section provides background information to support the comprehension of our study. Section 2.1 briefly explains code clones, including definition and types of code clones. Section 2.2 discusses some common techniques to support code clone detection. Section 2.3 presents a technique for code clone detection based on similar sequences of method calls.

2.1. Code Clones

Code clones are fragments of source code that are similar, or exactly the same, with respect to structure or programming logic [Rattan, 2013]. Eventually, software developers clone code, i.e., copy and paste existing code from one part to another of a software system, to support the development of new system features [Rattan, 2013]. One reason for this practice is that code clones contains tested components, clear solutions, and useful programming logic that may be reused [Ducasse et al., 1999]. In addition, some development external factors, such as time constraints and productivity evaluation, may lead to code cloning practices [Ducasse et al., 1999].

Two code fragments may be considered code clones based on syntax similarity or also based on features provided by the fragments [Koschke et al., 2006]. Roy et al. [2009] discuss four main types of code clones as follows. Code clone *Type I* is related to exact copies of source code fragments, without modifications other than blank spaces and comments. *Type II* is defined as identical copies with respect to syntax. In this case, modifications cover variables, types, and function identifiers, for instance. Code clone *Type III* concerns copies with more significant modifications, such as addition, editing, and removal of declarations. Finally, *Type IV* is related to two fragments of source code that have different structures, but provide the same computation. This type is considered the most complex because it requires an understanding of the code behavior to be detected [Bellon et al., 2007].

2.2. Common Techniques for Code Clone Detection

Many methods have been proposed in the literature to support the detection of code clones, using different detection strategies [Bellon et al., 2007; Hummel et al., 2010; Johnson, 1993]. Depending on the type of code clone we aim to detect, some of these strategies may provide better results than others. Some of the main strategies in literature are text-based, token-based, tree-based, and PDG-based [Rattan et al., 2013].

The text-based methods aim to detect code clones that differ at most in terms of code format layout and comments [Johnson, 1993]. Some transformation in the source code layout may be conducted to ease the textual analysis. In the token-based detection of code clones, tokens are extracted for the source code lines under analysis, in accordance with the analyzed programming language [Yuan and Guo, 2012]. Blank spaces, line breaks, and comments are removed for the comparison of tokens. Tree-based methods use Abstract Syntax Trees (AST) to support code clone detection [Baxter et al., 1998]. Identical or similar sub-trees are detected by a pairwise comparison. Finally, code clones can also be detected by means of the Program Dependency Graph (PDG) [Krinke, 2001]. Basically, a PDG is a representation of data and control dependencies in a software system. Subgraphs are analyzed for detection of code clones.

2.3. Code Clone Detection based on Sequence of Method Calls

In previous work [Paiva, 2016], Paiva propose a technique to detect code clones based on similar sequences of method calls. This technique analyses all method calls from a given software system. The occurrence of similar sequences of method calls in two different parts of the source code indicates a possible occurrence of code clone. The longer the sequence, the greater are the chances of these two parts being a code clone. The reason behind this technique is that, when source code fragments are copied from one part to another in the software system, these copies carry a set of instructions such as declarations, statements, operations, and method calls, for instance [Kim et al., 2004]. Therefore, code clones may be considered as replications of instruction sets in different parts of the system.

This code clone detection technique can be summarized in four consecutive steps. In Step 1, given an input software system, we extract all method calls from the system. Then, in Step 2, we gather the extracted method calls per method. In Step 3, we select only methods with at least 10[1] method calls. In Step 4, for each selected method, we pairwise the calls of this method with other methods (in the same file or in other files from the system) to assess whether they have similar sequences of calls. If the two compared methods have similar sequences of method calls, we consider both sequences as a code clone candidate.

[1] We used 10 as default in this study based on empirical observations with varying values [Paiva, 2016].

3. Study Settings

This paper aims to evaluate the technique proposed by Paiva [Paiva, 2016], to detect code clones based on similar sequences of method calls (Section 2.3). In this evaluation, we executed a survey with 25 participants. Each participant had to analyze 12 code clone candidates detected using the technique under evaluation.

Background of the Participants. The 25 participants of this survey are undergraduate and graduate students. Each participant filled in a characterization form[2] intended to collect profile data. The form questions asked the participants regarding their skills on the following areas: Object Oriented Programming, Java Programming, Bad Smells, Code Clone, and Work Experience. The characterization form asked the subjects to self-classify their skills on these areas using the following options: None, Few, Moderate, or Expert. The goal of collecting profile data is to further correlate with results of the main survey.

Survey Structure. The main survey was submitted to the participants via a specialized website[3]. Each subject was invited to analyze and decide for 12 pairs of code if they are clones. These 12 code clone candidates were randomly chosen from the total amount of 187 code clones automatically detected in 5 software systems using the technique presented in Section 2.3. Participants of this study rely on their subjective opinion to decide whether a pair of code fragment is clone. In addition, they were also asked to fill in a free text field explaining the reason of their choice. The same set of code clone candidates was presented to all subjects.

Training Session and Guidelines. All subjects received a 30-minute training session, divided in two parts. The first part was an explanation about refactoring code clones [Fowler, 1999]. For instance, Extracted Method was used to exemplify a recommended refactoring for code clone. In the second part of the training, we give general guidelines to answer the survey. The code clones of the training were not the ones of the applied survey. Both training and the survey took place in a controlled environment; i.e., a computer laboratory with 30 equally personal computers. After the training session, participants had one hour to finish the survey. This time was enough for all.

4. Results and Discussion

This section presents and discusses the results of our survey. Section 4.1 characterizes the participants of this survey with respect to five areas of expertise. Section 4.2 summarizes the main results on whether participants confirm the code clones detected by the evaluated technique. Section 4.3 discusses the results considering the different profiles of participants, with focus on their knowledge in code clone.

4.1. Characterization of Participants

Figure 1 shows the profiles of all 25 participants of survey according to five knowledge areas. For simplification purpose, we classify the knowledge of participants in two

[2] https://eSurv.org?u=characterizationform

[3] https://eSurv.org/?u=iscodeclone

groups: few/none and moderate/expert. The y-axis presents the absolute number of participants per knowledge level. Data in Figure 1 show that, in general, participants have balance experience in the asked areas. For instance, most participants consider themselves as moderate or expert in object-oriented programming (OOP), but with few or no experience in other areas. These results are somehow expected since participants are mostly undergraduate students.

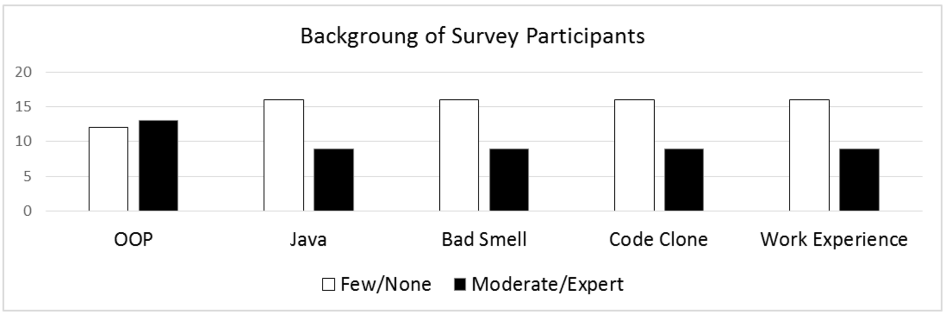

Figure 1. Background of the participants.

4.2. Overall Results

Figure 2 shows the overall agreement of participants in each code clone candidate. An agreement of 100% means that all participants indicated the pair of fragments as a code clone. This figure is ordered from lowest to highest agreement. That is, the first column in Figure 2 is not necessarily the first clone candidate in the survey. Figure 2 shows that, for seven candidates (i.e., columns 6 to 12), more than 90% of participants agree that they are actual code clones. In 3 out of 12 cases, between 50% and 60% of participants also agree with the codes as being clones. On the opposite, only two cases have agreement below 50%. Therefore, in general, results in Figure 2 indicate that a coincident sequence of method calls can be a good indicative of code clone.

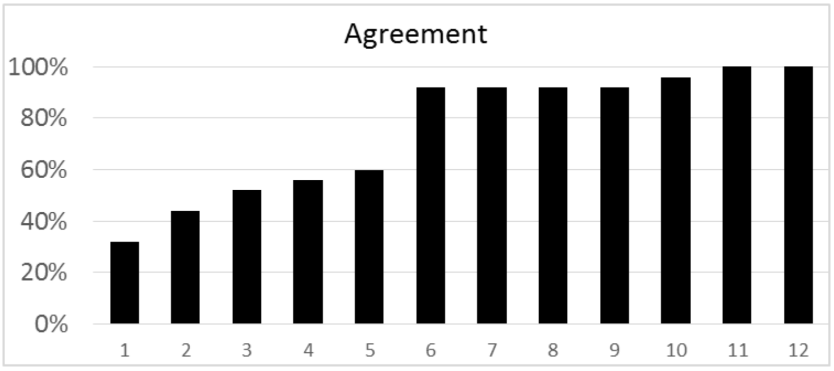

Figure 2. Agreement of the code clone candidates.

We observed in the qualitative answers that divergent judgment express how difficult is to have an agreement. Even for cases with low agreement, the open answers make it clear that the judgement of code clones is not easy. For instance, we present below two opposite points of view expressed by two participants with respect to the same code clone candidate. These participants disagree since the former considers both codes as doing same computation, while the latter does not.

"Yes, it is code clone. Although they are different from each other, codes do the same thing."

"No, it is not code clone. They are similar, but the methods do very different calculations."

4.2. Analysis of Participant Profiles

We investigate a possible relation between the background of participants and their tendency to indicate a code fragment clone or not. A general interesting observation is that, for most code fragments, participants with weak background – i.e., with the none or few knowledge in the asked expertise (see Section 3) – tend to agree with the code clone detection more often. As a representative of this observation, Figure 3 shows the results for all participants classified by their claimed expertise in code clone. Since the number of participants with each profile varies considerably, we present results as a percentage of participants with the specific profile. For instance, considering case 1 in Figure 3, almost 40% of participants with no or few experience in code clone indicated this pair of fragments as code clone.

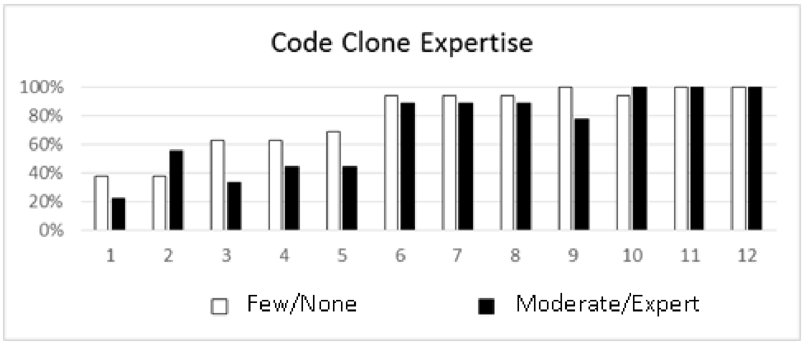

Figure 3. Background of the participants.

As presented in Figure 3, only in two cases, namely 2 and 10, a higher percentage of participants with moderate or high expertise in code clone confirmed the code clone candidates in comparison with participants in the few/none category. On the other hand, in eight cases, the percentage of participants with low profile are higher than the percentage of participants with better knowledge. Due to space constraints, we do not present the other analyzed profiles, but this general observation is similar to most cases of participants considering expertise in OOP, Java, and Bad Smell. Unlike the other profiles, participants with higher work experience confirmed more code clone candidates than participants with low work experience did.

5. Threats to Validity

Since the survey involves several steps, the various threats may have impacted on its validity [Wohlin et al., 2012]. The main question is whether the results can be generalized to a broader population. We cannot claim that our findings can directly apply to other software systems and to different clone detection techniques. However, we randomly selected code clone candidates for industry-strength projects and, so, we believe the most of our findings also hold to other similar contexts.

Another threat to generalization is that participants of our study are undergraduate and post-graduate students. However, a recent work have shown that the results of experiments with students are similar to the results with experienced professionals [Salman et al., 2015]. In addition, most of our results depend on the subjective opinion of the selected participants. Finally, the paper conclusions were derived based on a discussion among the paper authors, and other researchers may come up with different a point of view. To easy replications and further analysis, we report the complete data in a supplementary website[4].

6. Related Work

Several techniques have been proposed to detect code clone [Bellon et al., 2007; Hummel et al., 2010; Johnson, 1993]. These detection techniques are also evaluated in published research work [Bellon et al., 2007; Roy et al., 2009]. For instance, Kim et al (2004) conducted an ethnographic study in order to understand programmers copy and paste practices. Based on their analysis, the authors constructed a taxonomy of code clone patterns. Bellon et al. (2007) performed an experiment to evaluate six techniques to detect code clone. The code clone candidates were evaluated by one of the authors as independent third party. Unlike Bellon et al., we rely on 25 undergraduate and graduate students to evaluate the code clone candidates.

7. Conclusions and Future Work

Many techniques for detecting code clones have been proposed in the past [Rattan et al, 2013]. However, it is hard to validate code clone candidates automatically detected by these techniques. This paper described the survey for evaluation of code clones detected using similar sequences of method calls [Paiva, 2016]. Twenty five participants with different developer profiles answered this survey. After a 30-minute training session, participants had one hour to confirm 12 code clone candidates randomly mined from 5 software systems.

The survey results indicate that more than 90% of participants confirmed the identified code clones. We also investigated the impact of participant profiles on their answers. In this sense, we observed that participants with weak background tend to confirm more code clones than participants with stronger background. However, the opposite happened with respect to work experience. That is, participants with more work experience confirmed more code clones than participants with low experience.

For future work, we plan to replicate this survey with more participants, focusing on different detection techniques and varying types of code clone candidates (e.g., types I, II, III, and IV, see Section 2.1). Based on the results of this study, we also plan to improve the technique to detect code clones based on similar sequences of method calls.

About the paper title: Dolly and Shaun are famous sheep. One of them is a clone.

References

Baxter, I. D., Yahin, A., Moura, L., SantAnna, M., and Bier, L. (1998). Clone Detection using Abstract Syntax Trees. In Proc. of Int'l Conference on Software Maintenance (ICSM).

Bellon, S., Koschke, R., Antoniol, G., Krinke, J., Merlo, E. (2007) Comparison and Evaluation of Clone Detection Tools. Transactions on Software Engineering (TSE), 33(9), pp. 577-591.

Ducasse, S., Rieger, M., and Demeyer, S. (1999). A Language Independent Approach for Detecting Duplicated Code. International Conference on Software Maintenance (ICSM).

[4] http://ampaiva.github.io/mcsheep/

Fowler, M. (1999). Refactoring: Improving the Design of Existing Code. Addison Wesley.

Gode, N. and Koschke, R. (2011). Frequency and Risks of Changes to Clones. In Proc. of International Conference Software Engineering (ICSE), pp. 311-320.

Hummel, B., Juergens, E., Heinemann, L., and Conradt, M. (2010). Index-based Code Clone Detection: Incremental, Distributed, Scalable. In Proc. of Int'l Conf. on Software Maintenance (ICSM), pp. 1-9.

Johnson, J. H. (1993). Identifying Redundancy in Source Code using Fingerprints. In Proc. of the Conference of the Centre for Advanced Studies on Collaborative Research (CASCON), pp. 171-183.

Kim, M., Bergman, L., Lau, T., Notkin, D. (2004) An Ethnographic Study of Copy and Paste Programming Practices in OOPL. In Proc. of Int'l Symp. on Empirical Soft. Eng. (ISESE), pp. 83-92.

Kontogiannis, K. (1997). Evaluation Experiments on the Detection of Programming Patterns using Software Metrics. In Proc. of Working Conference on Reverse Engineering (WCRE), pp. 44-54.

Komondoor, R. and Horwitz, S. (2001). Using Slicing to Identify Duplication in Source Code. In Proc. of International Static Analysis Symposium (SAS), pp. 40-56.

Koschke, R., Falke, R., and Frenzel, P. (2006) Clone Detection Using Abstract Syntax Suffix Trees. In Proc. of the 13th Working Conference on Reverse Engineering (WCRE), pp. 253-262.

Krinke, J. (2001). Identifying Similar Code with Program Dependence Graphs. In Proc. of the Working Conference on Reverse Engineering (WCRE), pp. 301-309.

Marinescu, R. (2004). Detection Strategies: Metrics-based Rules for Detecting Design Flaws. In Proc. of International Conference on Software Maintenance (ICSM), pp. 350-359.

Paiva, A. (2016). On the Detection of Code Clone with Sequence of Method Calls. Master Dissertation, Federal University of Minas Gerais (UFMG).

Rattan, D., Bhatia, R., and Singh, M. (2013). Software Clone Detection: a Systematic Review. Information and Software Technology (IST).

Roy, C., Cordy, J., Koschke, R. (2009) Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach. Science of Computer Programming, 74(7), pp. 470-495.

Salman, I., Misirli, A., and Juristo, N. (2015). Are Students Representatives of Professionals in Software Engineering Experiments? In Proc. of Int'l Conference on Software Engineering (ICSE), pp. 666–676.

Wahler, V., Seipel, D., Wolff, J., and Fischer, G. (2004). Clone Detection in Source Code by Frequent Itemset Techniques. In Proc. of Int'l Workshop on Source Code Analysis and Manipulation (SCAM).

Wohlin, C. et al. (2012). Experimentation in software engineering. Springer.

Yuan, Y. and Guo, Y. (2012) Boreas: An Accurate and Scalable Token-based Approach to Code Clone Detection. In Proc. of Int'l Conf. on Automated Software Engineering (ASE).