ADV MATH SCI JOURNAL
Advances in Mathematics: Scientific Journal **9** (2020), no.6, 3783–3788
ISSN: 1857-8365 (printed); 1857-8438 (electronic)
https://doi.org/10.37418/amsj.9.6.56 Spec Issiue on ICAML-2020

TOWARDS AN APPROACH TO RECOMMEND AND RANK CLONES FOR REFACTORING

M. KAUR[1] AND D. RATTAN

ABSTRACT. Management of code clones has become a popular research area in recent years. Clones can be managed through tracking and refactoring. But there exists a gap between clone detection and clone refactoring due to lack of techniques which filter important clones for refactoring from large clone detection results. Since it is not feasible to manually check the suitability of each detected clone for applying refactoring, it is required to filter clones suitable for refactoring. In this paper, we proposed an approach to filter and rank important clones for refactoring. The approach is based on set of features that can be extracted from cloned code and clone evolution study.

1. INTRODUCTION

A software can contain duplicate code fragments in same source code file or in different source code files. Existence of such duplicate fragments are called code clones, [7]. Developers copy existing source code from one place and paste it at different places with or without changes. There can be many reasons, [7], behind such behavior of developer.

Presence of clones can increase maintenance cost, [1]. If there is a bug in cloned fragment, then during corrective maintenance, the bug needs to be removed from all the copies of cloned fragment. If developer missed change in

[1]*corresponding author*

2010 *Mathematics Subject Classification.* 68N30, 68T35.

Key words and phrases. software clone, clone refactoring, clone recommendation, clone maintenance.

any one cloned fragment, it will lead to bug propagation, [8]. So, detection of clones from software is very much required during software maintenance phase.

There exists many clone detection tools, [9], like CCFinderX, ConQat, Nicad etc. which find clones from source code written in different languages like Java, C, C++ and Python. These tools return clone results in different formats. To use these results in maintenance related task is also challenging. Firstly, all detected clones are not suitable for every type of maintenance task. Some can be removed using suitable refactoring technique and others may not be feasible to refactor and presence of such clones need to be tracked so that if one clone changes, similar copies of that clone also get changed. Second problem to use results of clone detection tools is that it is difficult for maintainer to identify subset of clones from existing clones that are suitable for refactoring or tracking. Here, maintainer need to evaluate each clone manually to check its suitability whether it can be refactored or tracked. Third problem is related to set priority of clone maintenance task. Clones which contain bug need to refactored first as compared to other clones. So, there is a need to rank clones so that critical clones in terms of maintenance can be handled on priority. The current paper focus on solution for problems mentioned above.

The rest of the paper is organized as follows. The section 2 gives the detail of solution that focus on filtering suitable clones for clone maintenance related tasks. The related work has been discussed in Section 3. Section 4 concludes the paper.

2. Proposed Method

Large number of techniques have been developed to detect clones from software system. Tools are freely available to detect clones from desired source code. But detecting clones is not enough, management of clones is also required. Clones can be removed from source code through refactoring. But it is not feasible to examine each detected clone manually to test whether it is suitable for refactoring or not from large clone detection results reported by a clone detection tool. The work of maintainer can become easy if he can have subset of detected clones on which refactoring can be applied. Our current approach aims to develop a system which can filter important clones for refactoring and also assign rank to clones so that most critical clones can be managed on priority. The following subsections discussed the proposed approach.

2.1. **Clone detection.** Detection of clones from source code is the prime task that need to be done for managing clones. Since many tools are available for clone detection, we are using the existing tools for clone detection. Literature review of clone refactoring depicts that most of research [3,4,6] related to clone refactoring has used the tool CCFinderX and NICAD. Our current approach will also use the same tools for clone detection. CCFinderX is token based tool and it can detect Type-1 and Type-2 clones whereas NICAD is text-based tool and can detect Type-1, Type-2 and Type-3 clones. Both tools have more than 90% recall for Type-1, Type-2 and Type-3 clones, [9].

2.2. **Extract clone features.** The second step is to extract the features of clones that can help to filter clones which are important for refactoring. Table 1 lists the feature of a clone [11–13] that can be extracted.

Table 1: Clone Features

Feature Name	**Explanation**
Clone Size	Size of clone in LOC. It helps to know whether clone is small or large. More the size of clone, more it can be beneficial for clone management.
Type of clone statement	Identify whether clone contains loop, conditional statement or arithmetic statement.
Clone location	Specify whether the clone fragments of a clone group exist in same method, same file or same class hierarchy.
Clone Type	Tell about whether clone is Type-1, Type-2, Type-3 and Type-4 clone.
Cyclomatic Complexity of cloned code.	Defines the complexity of the program. More the value of cyclomatic complexity, more are the chances to refactor the code.
Size of Clone Class	Size of clone class specifies the number of clone fragments in the clone class. More the size of clone class, more clone fragments will be refactored.
Life expectancy of clone	If clones are short lived then such clones do not require immediate refactoring.
Bug-Proneness of a clone	If a clone contain bug then it needs to be refactored on priority.
Frequency of clone Change	Specifies the change-proneness of a clone. If a clone is changing very frequently, it can be refactored to avoid inconsistent changes to clones.
Clone Age	It specifies how long the clone exists in the software. As the clone becomes older, it becomes more stable and therefore, requires less refactoring.
Frequency of file changes containing clone	If a file containing clone is changing very frequently then higher are the chances of inconsistent changes to clones. So, refactoring can be applied to remove clones.
Size of cloned code out of total size of a method	If a cloned code is contained in a method and there is more size of the cloned code than non-cloned code. Then, it is easier to apply refactoring.
Percentage of method calls in cloned code	If cloned code contains method calls, then it is difficult to refactor clone.

Number of referenced variables in a cloned code	More external variables are referenced in a cloned code, more will be the dependency of code on external variables and it requires more effort to refactor the clone.
Percentage of differences in clone peers of a clone group	If clone fragments of a clone class have large number of differences, then more refactoring effort is required to remove clones.

2.3. **Calculate Maintenance Overhead.** Maintenance overhead is based on the values of features extracted from cloned code. Maintenance Overhead for a clone can be calculated after assigning weightage to features extracted in previous step. A weight assigned to a feature depends on the role of feature in maintainace related task.

2.4. **Assign Rank to clones.** Ranks are assigned to each clone based on the value of Maintenance overhead. The clone with highest maintenance overhead is assigned highest rank so that it can be refactored on priority.

3. Related Work

Choi et al. in [2] proposed a method to extract clones from the output of clone detection tool, CCFinder, for refactoring. The method selects those clone sets for refactoring having higher combined clone metrics values. Tairas and Gray in [10] unified the process of clone detection and refactoring. Their tool CeDAR takes input from clone detection tools and filter the group of clones which are suitable for extract method refactoring. To filter clones, the pre-conditions are checked which are required to be fulfilled before apply extract method refactoring.

Higo et al. in [4] developed refactoring support tool, Aries, for removing code clones. It uses the tool CCFinder to detect clones from software. Fontana et al. in [3] developed tool, DCRA, that suggests the refactoring type that can be applied to a clone pair so that duplication can be removed. The tool uses NICAD to detect clones.

Clone refactoring tool, Jdeodorant in [6], can import the clone detection results of CCFinder, NICAD, ConQat, CloneDR and Deckard. It supports Extract Method, Pull-Up Method and Extract Utility Method refactoring techniques to remove clones from source code. Tool SPCP-Miner, [5], identifies clones that are important candidates for refactoring and tracking among large set of detected

clones from a software and also rank these important clones according to their priority in fixing through refactoring or tracking.

Venkatasubramanyam et al. in [11] proposed an approach to prioritize code clones to find the order of fixing of clones. They considered several factors like clone maintenance overhead, bug-proneness of the clone and refactoring magnitude for prioritizing clones for management related tasks.

Wang and Godfrey in [12] proposed an approach to recommend clones for refactoring by training decision tree based classifier. The classifier was trained from features of source code, context and history of clones. The results show that clone recommendation for refactoring can be done with high precision by learning from features of clones.

Yue et al. in [13] developed learning based approach that recommends clones for Extract method refactoring of clones. The approach is based on 34 features that are extracted from current status and past history of software. The results show that their approach recommends refactoring with 83% F-score within-project and 76% F-score cross-project settings.

4. Conclusion

In this paper, we reviewed studies related to clone refactoring and proposed an approach that can be used to filter important clones for refactoring from large clone detection results returned by clone detection tool. Clones can be assigned ranks so that critical clones can be given higher priority during maintenance tasks.

References

[1] D. Chatterji, J. C. Carver, N. A. Kraft, J. Harder: *Effects of cloned code on software maintainability: A replicated developer study*, Proc. Work. Conf. Reve. Engi., WCRE (2013), 112–121.

[2] E. Choi, N. Yoshida, T. Ishio, K. Inoue, T. Sano: *Extracting code clones for refactoring using combinations of clone metrics*, Proc. International Workshop on Software Clones, IWCS (2011), 7–13.

[3] F. A. Fontana, M. Zanoni, F. Zanoni: *A duplicated code refactoring advisor*, Proc. Int. Conference on Agile Software Development, (2015), 3-14.

[4] Y. HIGO, S. KUSUMOTO, K. INOUE: *A metric-based approach to identifying refactoring opportunities for merging code clones in a Java software system*, J. Softw. Maint. Evol. Res. Pract., **56**(6) (2008), 435-461.

[5] M. MONDAL, C. K. ROY, K. A. SCHNEIDER: *SPCP-Miner: A tool for mining code clones that are important for refactoring or tracking*, Proc. Int. Conf. Soft. Anal.,Evol., and Reeng., SANER (2015), 484–488.

[6] D. MAZINANIAN, N. TSANTALIS, R. STEIN, Z. VALENTA: *JDeodorant: Clone Refactoring*, Proc. International Conference on Software Engineering Companion, (2016), 613–616.

[7] D. RATTAN, R. BHATIA, M. SINGH: *Software clone detection: A systematic review*, Inf. Softw. Technol., **55**(7) (2013), 1165–1199.

[8] D. RATTAN, R. BHATIA, M. SINGH: *Detecting High Level Similarities in Source Code and Beyond*, International Journal of Energy, Information and Communications, **6**(2) (2015), 1–16.

[9] J. SVAJLENKO, C. K. ROY: *Evaluating clone detection tools with BigCloneBench*, Proc. Int. Conf. Softw. Maint. Evol., ICSME (2015), 131–140.

[10] R. TAIRAS, J. GRAY: *Increasing clone maintenance support by unifying clone detection and refactoring activities*, Inf. Softw. Technol., **54**(12) (2012), 1297–1307.

[11] R. D. VENKATASUBRAMANYAM, S. GUPTA, H. K. SINGH: *Prioritizing code clone detection results for clone managemen*, Proc. International Workshop on Software Clones, IWSC (2013), 30–36.

[12] W. WANG, M. W. GODFREY: *Recommending clones for refactoring using design, context, and history*, Proc. Int. Conf. Softw. Maint. Evol., ICSME (2014), 331–340.

[13] R. YUE, Z. GAO, N. MENG, Y. XIONG, X. WANG, J. D. MORGENTHALER: *Automatic clone recommendation for refactoring based on the present and the past*, Proc. Int. Conf. Softw. Maint. Evol., ICSME (2018), 115–126.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
BABA BANDA SINGH BAHADUR ENGINEERING COLLEGE
FATEHGARH SAHIB, PUNJAB, INDIA
Email address: manpreet.kaur@bbsbec.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
PUNJABI UNIVERSITY
PATIALA, PUNJAB, INDIA
Email address: dhavleesh@gmail.com