DRCC: an AST-Based Tool to Detect and Refactor Clone Code

Cong Wang
School of Software, Tsinghua Univ.
Beijing, China +86 15501257110
wangcong15@mails.tsinghua.edu.cn

***Abstract*—Refactoring, which software programmers apply to improve the maintainability, readability and simplicity of programs, is the process of restructuring current code of software projects. Although clone code, one of the most pervasive defects of code, has been largely studied in the literature, detection and refactoring can hardly reach the semantic level, which lead to the neglect of the clone code with reversed line orders. We present *DRCC*, which can detect and refactor clone code automatically by using the abstract syntax tree(AST). Besides, we propose an advanced definition of the similarity of different types of statement and design three significant algorithms. *DRCC* uses combination and extraction methods to refactor the clone Java code.**

1. Introduction

Refactoring, which software programmers apply to improve the maintainability, readability and simplicity of programs, is the process of restructuring current code of software projects. Clone code, one of the most pervasive defects of code, has been largely studied in the literature. When fixing bugs and adding new functions which are similar with existing snippets, programmers often copy and paste the code before.

Several methods of clone code detection have been put into use, such as tokens, lines, functions, characters, nodes in abstract syntax tree and so on.

Kamiya presents *CCFinder*[1] and proposes a clone detection technique, which consists of transformation of input source text and token-by-token comparison. *CCFinder*'s idea is to change the code into a regular form but not to change the meaning so that the token-based clone detection method can be scalable to large software systems and easily adaptable to many other programming language.

Baker presents *dup*[2], which searches code for all pairs of duplicated sections. *dup* is line-based because lines are considered to be identical if they contain the same sequence of characters after removing comments and white spaces. Baker only cares about the syntax level, so the semantics of the statements is not discussed in this paper. Data structures are maintained with lines to reduce the space requirement.

Mayrand proposes a metrics-based technique[3] which automatically identify duplicate and near duplicate functions. An ordinal scale of eight cloning levels is defined, which ranges from exact copy to distinct functions. Mayrand describes the metrics, the thresholds and the process used.

Ducasse proposes a visual approach[4] that is language independent and presents a tool that requires no parsing, yet is able to detect a significant amount of clone code. Language independence means that the source code must be transformed into an internal format, and a sophisticated comparison algorithm is needed to perform on the internal code. Ducasse uses string manipulation operations and string matching to finish this part.

Baxter presents a simple and practical technique[5] for detecting clones code over arbitrary program fragments in program source code by using abstract syntax trees. Baxter defines "clone" as a program snippet that identical to another one and defines "near miss clone" as that nearly identical to another one. In this paper, the similarity between two sub-trees is computed by 2S/(2S+L+R). "S" means the number of shared nodes. "L" and "R" mean the number of different nodes in sub-trees.

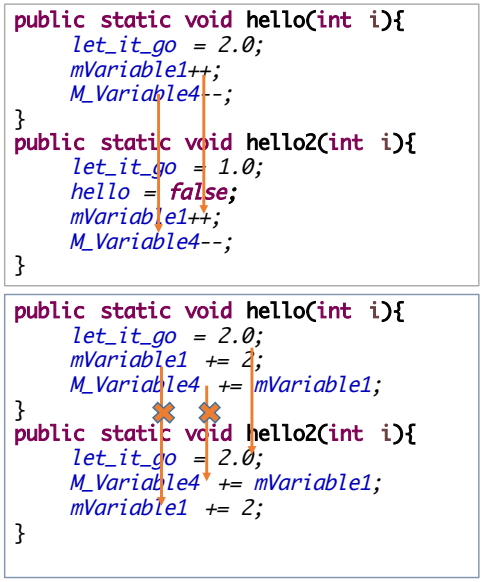

Figure 1. Clone code with reversed line orders

However, detection and refactoring can hardly reach the semantic level, which lead to the neglect of the clone code with reversed line orders. Two similar snippets in Figure 1

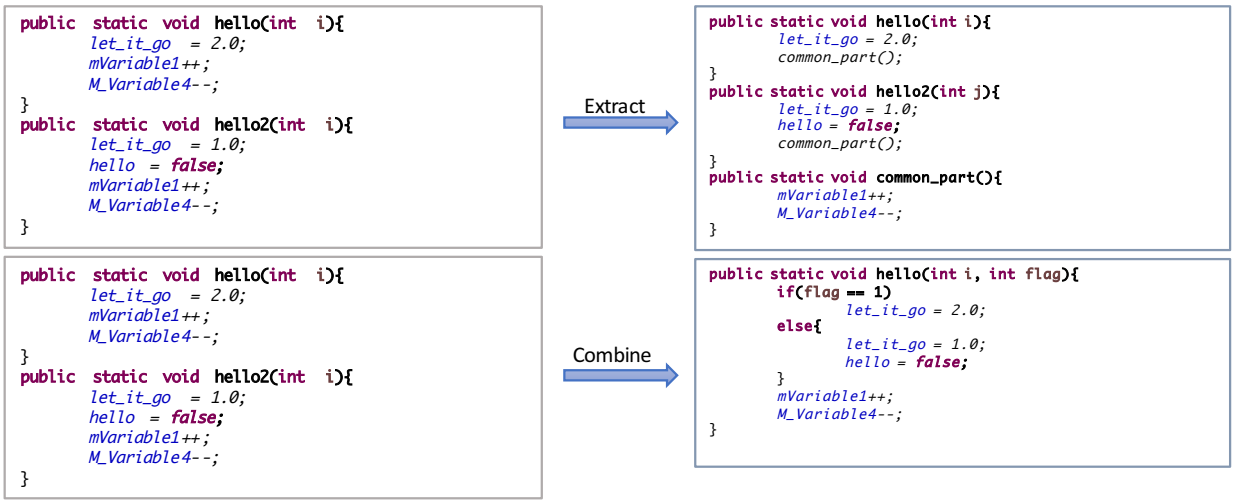

Figure 2. Refactoring methods

shows two different scenes of clone code. As far as the refactoring is concerned, the reversed line orders should be handled so that the similarity can be more appropriate. On the side of refactoring, methods of extraction and combination can be used together as the extraction holds the advantage of same code in functions and combination helps to merge functions.

For example, matches in the former comparison in Figure 1 don't have intersection but in the latter one the matches have influence on each other leading to the inaccuracy of similarity on this occasion, which is called the cross of match in this paper caused by the reversed line orders.

Examples of refactoring are shown in Figure 2. We use the methods of combination and extraction to finish the task of refactoring.

Contribution:

1) Propose a novel definition of the similarity that handles the problem of the cross of match.

2) Propose an algorithm refactoring clone code that preprocess the program with reversed line orders.

3) Present a tool to detect and refactor clone Java code in function and block level using abstract syntax tree.

We present the definition of Same Statements and Similarity in Section 2. After the basic definitions, Section 3 presents the main algorithms to detect and refactor clone code. When finishing the tool, we show some results of the work on some popular code. Then in Section 5 and the final, we make a conclusion and discuss about the future work.

2. Definition

To detect and refactor the clone code, the first step is to definite similarity between code snippets. The definition of same statements will be given in Section 2.1 and Similarity in Section 2.2.

2.1. Definition of Same Statements

Algorithm 1 Judge the Sameness of Two Statements

Input: Statement stmt1, Statement stmt2, Double threshold
Output: Boolean result-stmt1 and stmt2 are same

1: **if** $stmt1.type! = stmt2.type$ **then**
2: return false
3: **else if** $stmt1.type$ = PrimaryStmt **then**
4: **if** $stmt1 = stmt2$ **then**
5: return true
6: **else**
7: return false
8: **end if**
9: **else if** $stmt1.type$ = IfStmt **then**
10: **if** $stmt1.con = stmt2.con$ and $stmt1.elsStruct = stmt2.elsStruct$ **then**
11: return $(SimSub(stmt1, stmt2) > threshold)$
12: **else**
13: return false
14: **end if**
15: **else if** $stmt1.type$ = LoopStmt **then**
16: **if** $stmt1.con = stmt2.con$ and $stmt1.loopType = stmt2.loopType$ **then**
17: return $(SimSub(stmt1, stmt2) > threshold)$
18: **else**
19: return false
20: **end if**
21: **end if**

The definition of same statement is significant and basic for the detection of clone code. Through analysing the structure of the code by abstract syntax tree, the origin code is modeled into a tree with a large number of node, indicating that each statement, which is simple or complex, is one of the nodes having many of child nodes. The comparison between two statements can be converted into the analysis between the two nodes and their child nodes in the abstract syntax tree.

When dealing with two statements, the first step is to compare the type, including "Primary Statement"(PrimaryStmt), "If Statement"(IfStmt), "Loop Statement"(LoopStmt). Then, statements with the same type is analysed with their conditions, sub-blocks and structures. The definition is presented in Definition 1-4.

Definition 1. *Statements with different types is considered as different ones.*

Definition 2. *Statements a and b with the type PrimaryStmt are same statements when and only when a = b*

Definition 3. *Statements a and b with the type IfStmt are same statements when and only when the condition and structure of "else statement" of them are the same and the product of the similarities of sub-blocks is more than the threshold.*

Definition 4. *Statements a and b with the type LoopStmt are same statements when and only when the condition and type of "loop statement"(for, while, do...while) of them are the same and the similarity of the sub-block is more than the threshold.*

It is obvious in Definition 3 & 4 that the threshold is not only the parameter to detect clone code in function level, but an important number to judge the two statements as well.

The method to judge the sameness of two statements is shown in Algorithm 1. SimSub($stmt1$, $stmt2$) calculates the product of the similarities of sub-blocks. The definition of Similarity shall be given in section 2.2.

2.2. Definition of Similarity

Depended on the definition of same Statements, Similarity is defined as Definition 5. The same statements cannot have the cross of match. "S" means the number of nodes in the same statements, while "L" and "R" mean the number of nodes in different functions fc1 and fc2.

Definition 5. *Similarity = 2S / (2S + L + R)*

3. Algorithm

We have proposed the definition of Same Statements and Similarity. Then in this section, we presents three main algorithms:

1) Order the statements in Functions
2) Compute Similarity of Methods
3) Refactor Clone Code

3.1. Algorithm: Order the Statements in Functions

3.2. Algorithm: Compute Similarity of Methods

3.3. Algorithm: Refactor Clone Code

Algorithm 2 Order the Statements in Functions

Input: Function $inFunc$
Output: Function $outFunc$

```
outFunc = inFunc
Bk = outFunc.block
Dt = newDependenceTree()
FlagNumber = 0
for stmt in Bk.statements do
    Dt.add(stmt)
    if stmt.hasDependenceOnPrevious() then
        FlagNumber+ = 1
    end if
    stmt.setFlagNumber(FlagNumber)
end for
outFunc.reorderStatements()
return outFunc
```

Algorithm 3 Compute Similarity of Methods

Input: Function $fc1$, Function $fc2$
Output: double $result$-similarity of $fc1$ and $fc2$

```
totalNode = fc1.totalNode + fc2.totalNode
List = sameStatementsMatched(fc1, fc2)
BestList = List.bestChoice()
bestMatchedNode = BestList.totalNode()
result = bestMatchedNode/totalNode
return result
```

4. Evaluation

5. Conclusion

In this paper, we have proposed a novel definition of similarity that handles the problem of the cross of match and proposed an algorithm refactoring clone code that preprocess the program with reversed line orders. Depending on the definition and the algorithm, we have presented DRCC- an AST-based tool to detect and refactor clone Java code in function, which uses the abstract syntax tree.

Future Work

On the one hand, our current work uses float number as threshold that is not visual enough. The ease of use can be improved if the user can decide the threshold by make choices on the sample code snippets. Maybe we can calculate all the possible clone code and present the in order

Algorithm 4 Refactor Clone code

Input: Class $javaIn$, double $threshold$, int $LenMin$
Output: Class $javaOut$

```
javaOut = javaIn
javaOut.extractSnippets(LenMin)
javaOut.combineMethods(threshold)
return javaOut
```

to the user so that the tool can get the expected threshold though users' choices.

On the other hand, today many software system are implemented in multi-languages so the tool can be expanded into other programming languages. For example, JavaScript, significant to be brief, is so widely used in websites that the detection and refactoring of JavaScript is important as well.

Acknowledgments

We want to thank Jiaguang Sun, Fei He and Pisa Song for guidance. This work has been helped by Dexi Wang, Xinrui Guo, Han Liu and Zuxing Gu for instructive discussions.

References

[1] Kamiya, Toshihiro, S. Kusumoto, and K. Inoue. "CCFinder: A Multilinguistic Token-Based Code Clone Detection System for Large Scale Source Code. IEEE Trans Softw Eng." IEEE Transactions on Software Engineering 28.7(2002):654-670.

[2] Baker, Brenda S. "A Program for Identifying Duplicated Code." Computing Science & Statistics (1992).

[3] Mayrand, Jean, C. Leblanc, and E. Merlo. "Experiment on the Automatic Detection of Function Clones in a Software System Using Metrics.." icsm IEEE Computer Society, 1996:244.

[4] Ducasse, St&#, et al. "A Language Independent Approach for Detecting Duplicated Code." 2012 28th IEEE International Conference on Software Maintenance (ICSM) IEEE Computer Society, 1999:109-109.

[5] Baxter, Ira D., et al. "Clone Detection Using Abstract Syntax Trees." icsm IEEE Computer Society, 1998:368.