© 2017 IJSRSET | Volume 3 | Issue 5 | Print ISSN: 2395-1990 | Online ISSN : 2394-4099
Themed Section: Engineering and Technology

Code Clone Detection based on Logical Similarity : A Review

Ashish N. Runwal*, Akash D. Waghmare
Department of Computer Engineering, SSBT COET, Jalgaon, Maharashtra, India

ABSTRACT

Code Clones are the entities in software ecosystems which can be unavoidable. Demand of software based clone detection has risen in industries day by day. Due to code duplication means the copy and paste activities, such pattern is recurrent thereby developers can reduce effort and time of rewriting similar code fragment by editing prewritten code. Code duplication may affect on quality, consistency, maintainability and comprehensibility. Here the trial is variety of syntax, compiler dependent language, and various coding patterns to resolve a single problem. There is lots of software tools, code clone detection algorithms exist, but they have some restrictions to detect perfect cloning. Earlier research and tools developed till now can find only Type-I, Type-II and some part of Type-III clones. Some tools are very slow and time consuming for comparing codes and with low in precision. Type-IV clone detection represents a challenge in current scenario. Type-IV is the Code with similar functionality that may be syntactically different but logically similar referred as semantic clones. This paper presents an algorithm for clone detection based on comparing parts of abstract syntax tree of programs and finding semantic coding styles.
Keywords: Code Clone, Type-I, Type-II, Type-III, Type-IV, Code duplication.

I. INTRODUCTION

The software maintenance is a complex task. When the code is large difficulty mainly occurs. The software code can have presence of large quantity of similar code fragments which increase the code length and maintenance cost. It is necessary to correct the error in corresponding cloned code fragments if any fault is occurs in some code. The detection and correction in huge software is difficult and complex task. Code clones are the similar code content that is knowingly or accidently occurs in code development process. Latest work states that clones are dangerous as formerly assumed and might upturn throughput. Clones do harm and provide undesired impact on software maintainability, also cloned code may be less error prone than non-cloned code. Code clones are section of source code that is doubled in numerous localities due to replica or even mirroring activity, or content copy paste by program writer. Clones are similar in syntax and logical way that have similar outcomes. Basically, clones have the functional similarity. Clone upsurges maintenance budget of software system, major standard methodologies to categorize the code clone in account that brings four benchmarks.

A. **Categories of code clones:**

1) Type-I Exact clones: Same code blocks except for differences in whitespace, design, and developer comments.

2) Type-II Renamed clones: Code with similar functionality and also syntactically identical fragment. Modification is made in replicated code, such as renaming identifiers.

3) Type-III Gapped clones: Code with Type-I and Type-II with additional insertion or deletion of statements are referenced as gapped clones.

4) Type-IV Semantic or logical clones: Code that may be syntactically different but logically similar referred as semantic clones.

Code clone detection is essential in order to utilize storage resources, maintain software and improve code productivity. Programming size increases for no reason. Code replication increases the overhead software maintenance, since bug introduction in the source may be replicated accidently or unknowingly. There are various code clone detection techniques for detecting such replicated codes, now introducing various detection techniques:

- Text-based
- Token-based
- AST-based (Abstract Syntax Tree)
- Dependency-graph-based
- Metrics-based
- Hybrid-based

A clone detector must compare all the possible code with each other. But such a process is very expensive and due to this it is supported by various metrics reducing unnecessary comparisons. In this paper we introduce our deal with syntactically different but logically similar code clones. Since us works with abstract syntax trees, it is believed to be a contribution to AST-based detection techniques. In the paper we describe our survey on the clone detection techniques based on AST.

I. RELATED WORK

The Code clone detection is now-a-days very popular domain for study. It is used in the number of types. There are 3 main type involved in the Code clone detection as Type I, Type II and Type III.

Keivanloo et al., in, [1] in this paper k-means clustering algorithm to replace the threshold-based cutoff step in the clone detection process is proposed. Use of k-means to determine the number of expected clusters as part of the configuration. The experimental result shows that use of k-means algorithm improves the performance significantly by 12%.

Toshihiro Kamiya, in, [2] in this paper a code-clone detection and its analysis method, based on an execution-semantic where two code fragments are defined to be identical to each other when their execution sequences include the same method invocations in the same order, in any possible executions of the target program and arbitrary granularity model of code fragments is presented. These experiments show that code-clone detection and its analysis method suitable for programming languages.

The ideas of clone-detection method are:

- Detection of similar sequences in an execution trace (dynamic information) of a target program and mapping of such similar sequences onto its source code (static artifact).
- Application of a frequent item-set algorithm to the arbitrary-granularity model (nodes at any depths in a sub tree of a call tree) to detect a type-3 code clone.

In an execution trace, type information is extracted from values stored in variables A call relation has been determined as a call tree itself. Also, control statements have been expanded as the results of dispatching of procedures.

Joshi et al., in, [3] in this paper multi-model learning technique to detect various types of code clone is proposed. Experiment shows that efforts of comparing the code line by line between two files are eliminated.

CCfinder tool is token based approach, detects sequence of source code and output. Tokens are removed, added or changed based on transformation rules that targets at regulation of identifiers and identification of structures. Then each identifier related to types, variables & constraints is changed with special token. CCfinder has no Graphical User Interface and the character based output is generated. The reorganization is confusing on only character based output information.

In token based clone discovery, all the source code is transformed into a development of symbol and direct recommendations. It is then inspected to classify replication subsequences.

Chodarev et al., in, [4] In this paper pattern recognition algorithm for clone detection based on comparing parts of abstract syntax tree of programs and finding repeating patterns is proposed. Implementation of the clone detection tool for Haskell was divided into two main parts:

1) Haskell parser producing abstract syntax trees of the source code,

2) Pattern analyzer recognizing the AST to find possible clones.

Both parts are implemented as separate programs that communicate using XML format as shown in fig. 1. Results of proposed algorithm found promising even on large code bases.

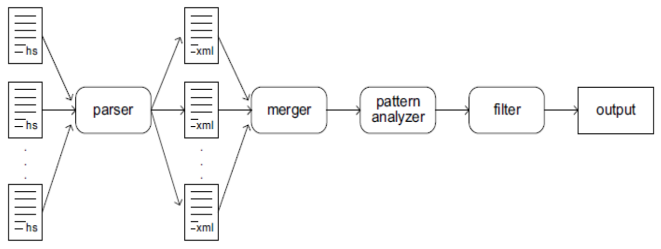

Figure 1. Architecture of Haskell clone detection tool

Nappa et al., in, [5] in this paper the first systematic study of patch deployment in client-side vulnerabilities is presented. These experiment shows that the median fraction of vulnerable hosts patched when exploits are released is at most 14%.

Patil et al., in, [6] in this paper the code reduction and decentralized system with multiple smart nodes is implemented. The code reduction method designates clusters for faster detection of clones. These experiment evaluates here as code complexity, weighted graphs enhances precision in detection.

Siim Karus and Karl Kilgi, in, [7] in this paper a set of wavelets-based code clone detection approach for detecting code clones is proposed. The experimental evaluation shows that approach is able to effectively identify more clones than alternative algorithms.

II. PROPOSED SOLUTION

We have proposed a clone detection technique based on AST. To find clones in the AST, we need principal to compare each sub tree to each other sub tree in the AST. Because this approach would not scale, it is necessary to compare all sub trees within the same partition in a second step. This comparison is a tree match, where we use an inexact match based on a similarity metric. The similarity metric measures the fraction of common nodes of two trees. Cloned sub trees that are themselves part of a complete cloned sub tree are combined to larger clones. Special care is taken of chained nodes that represent sequences in order to find cloned subsequences.

A. Abstract Syntax Tree:

The AST-based technique yields syntactic clones. And the AST-based clone detection offers many additional advantages as already mentioned in the introduction also shown in fig. 2. Partitioning the sub trees in the first stage helps a lot; the comparison of sub trees in the same partition is still pair wise and hence requires quadratic time. Moreover, the AST nodes are visited many times both in the comparison within a partition and across partitions because the same node could occur in a sub tree subsumed by a larger clone contained in a different partition.

We assume however that the clone detection is part of a larger system and the AST is already available. It would be valuable to have an AST-based technique.

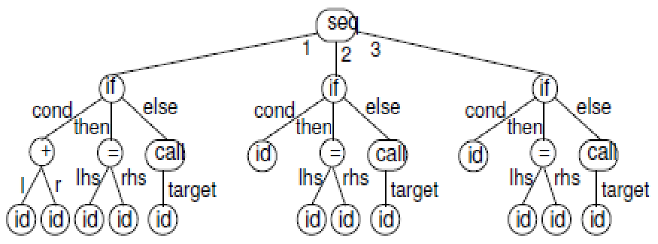

Figure 2. Examples AST

The codes which are syntactically different, but semantically similar remain hidden from code clone detection techniques. Abstract syntax tree can be used to overcome this challenge. Abstract syntax tree is a tree representation of the abstract syntactic structure of source code written in a programming language. By using abstract syntax tree, logically similar code clones can be detected.

III. PROBLEM STATEMENT

Code clone detection is a process of finding semantically and syntactically similar code clones. Code clones are code fragment similar to one another in the form of semantics and syntax. Code clones are created through mirroring activity or content copy paste. Finding semantically similar code clones is a challenging task in code clone detection. Due to this, the codes which are syntactically different, but semantically similar remain hidden from code clone detection techniques. Abstract syntax tree can be used to overcome this challenge. Abstract syntax tree is a tree representation of the abstract syntactic structure of source code written in a programming language. By using abstract syntax tree, logically similar code clones can be detected.

IV. CONCLUSION

From the discussion it is clear that code cloning detection is a technique of finding the semantically and syntactically similar clones. The fragment which is syntactically different, but semantically similar remains hidden from code clone detection techniques. Abstract syntax tree can be used to overcome this challenge. Usage of abstract syntax tree will give better and superior results than the traditional approach.

V. REFERENCES

[1] Iman Keivanloo, Feng Zhang, Ying Zou, 2015, “Threshold-Free Code Clone Detection for a Large Scale Heterogeneous Java Repository”, Saner IEEE.

[2] Toshihiro Kamiya, 2015, “An Execution-Semantic and Content and Context Based Code Clone Detection and Analysis”, IWSC IEEE.

[3] Ritesh V. Patil, Shashank D. Joshi, Sachin V. Shinde, V. Khanna, “An Effective Approach Using Dissimilarity Measures To Estimate Software Code Clone”,

[4] Sergej Chodarev, Emilia Pietrikova, Jan Kollar, 2015, “Haskell Clone Detection using Pattern Comparing Algorithm”, International Conference on Engineering of Modern Electric Systems, IEEE.

[5] Antonio Nappa, Richard Johnson, Leyla Bilge, Juan Caballero, Tudor Dumitras, 2015, “The Attack of the Clones: A Study of the Impact of Shared Code on Vulnerability Patching”, IEEE Symposium on Security and Privacy.

[6] Ritesh V. Patil, Shashank D. Joshi, Sachin V. Shinde, Digvijay A. Ajagekar, Shubham D. Banker, 2015, “Code Clone Detection Using Decentralised Architecture and Code Reduction”, International Conference on Pervasive Computing, IEEE.

[7] Siim Karus, Karl Kilgi, 2015, “Code Clone Detection using Wavelets”, IWSC IEEE.

[8] Rainer Koschke, Raimar Falke, Pierre Frenzel, 2006, “Clone Detection Using Abstract Syntax Suffix Trees”, 13th Working Conference on Reverse Engineering, IEEE.

[9] Tahira Khatoon, Priyansha Singh, Shikha Shukla, Dec. 2012, “Abstract Syntax Tree Based Clone Detection for Java Projects”, IOSR Journal of Engineering, Vol. 2, Issue 12.