Technique for Searching of Similar Code Segments

Akshat Agrawal, Sumit Kumar Yadav

Abstract— ***In this paper, we will be studying about various artifacts and constructs about many tools to help developer in their task of developing. These tools will try to fulfill the basic need of any developer which is to have similar code segments to help him to reduce his efforts. For this we have various tools available in market. After reading this paper the developer will be able to choose the best suitable code detection tool for his work.***

Index Terms— ***Artifacts, Code detection, Code segment, Constructs.***

I. INTRODUCTION

Today world is revolving around computer and internet. Basic need to drive any computer system is the hardware and its related software. Software is of many types and being classified according to the need of user who is going to be using it. Classification involves types such as

i. System software- Made to drive the system hardware or provide a platform for running application.

ii. Application software- These are software developed to ease the work of user.

iii. Programming software- These are created to support other program and application or to help developer in creating code.

The fundamental construct for creating software are some lines of code. These lines of code are referred as program by the developer. Developer can create a program from scratch or can make use of the existing code and perform some needed modification to fit to his requirement.

There are many available tools present in the market. These tools do differ from each other in many terms such as their way of representing code segments, their indexing technique, their code matching technique etc.

So in this paper we will provide a way for developer to find existing program and make use of them in their environment.

The rest of the paper is divided into four sections namely Motivation which state some previous work, Code detection tool classification describing type of tools, Code detection process describing the step by step methodology of tools, Scenario based example followed by Conclusion of the paper.

II. MOTIVATION

Collin McMillan, Mark Grechanik, Denys Poshyvanyk, Chen Fu, Qing Xie [1] developed a source code search engine that made use of three things for finding code as per user requirement. They made use of description of the application, API calls used by the application and data flows among those API calls to find source code [1]. Flow of data among various API calls was found to be a significant factor in finding code segments and was pioneered by them. Main objective was to semantically find out code rather than syntactically. Their result were find out to be most relevant with respect to other source code search engines such as Google Code and Sourceforge which are well known search engines for code due to semantic matching.

Mu-Woong Lee, Jong-Won Roh, Seung-won Hwang, Sunghun Kim [2] developed a instant code clone search tool which focused on improving performance by avoiding a post-mortem technique [2] followed by many tools. Usually tools finds the code after it has been developed but this tool tries to find code as it when developer is creating. They propose scalable indexing structures on vector abstractions of code. With this indexing approach they achieved a response time of sub-seconds.

Lingxiao Jiang, Ghassan Misherghi, Zhendong Su, Stephane Glondu [3] modeled a tree based technique for finding code clone. There algorithm focused on finding similar subtrees and characterized tree by numerical vector [3]. They also made use of clustering technique in which these numerical vectors are are clustered into one cluster with respect to Euclidean distance metric. This tool is able to work upon code created in C and JAVA.

III. CODE DETECTION TOOL CLASSIFICATION

In today industry, any development task is followed by review of some similar task being done before. So the developer is looking for the similar thing present in the nature, to make himself aware of its benefits and flaws. They make use of the existing structure. But in IT field this practice is hard to follow.

The reason behind this is that initially this way of developing code segment was not practiced. Developer try to create the code that would suit for only their application and as a result there is nothing available upon which one can extend. Also there was no efficient way to help developer search for similar code if needed.

There are two ways by which we can provide a developer to find related programs or code segments. First is to create a source code search engine which will be having similar functionality to that of search engines such as Google, but designed only for code. Second way is to build a tool precise for similar code detection.

So on the basis of analysis applied to the source code, the techniques can roughly be classified into four main categories: text based, token based, syntax based, and semantic based.

Manuscript received March, 2013.

Akshat Agrawal, Computer Science and Engineering, Lovely Professional University, Jalandhar, India,

Sumit Kumar Yadav, Computer Science and Engineering, Lovely Professional University, Jalandhar, India.

Text based approach make use of the source code or program without any modification to match them for the developer uses. They treat every word of the code with equal importance in the process of code matching. Johnson [4] was the first one to use text based matching and made use of the fingerprints on source code. Ducasse et al. [6] proposed a similar technique and made use of the dot plot. They considered a dot at (x, y) if these x and y (representing code) are equal. Similar Codes are identified as diagonal.

Token based approach was firstly used by Brenda Baker [7]. In this approach each line of code is first divided into fixed length word known as tokens. Similar codes are then found on the basis of matching percentage between these tokens. These tokens can be identifier and literals. Kamiya et al. [8] then later incorporated this technique in his code tool CCFinder.

Syntax based approach was first formulated by Baxter et al. in his tool namely CloneDr. In this technique code is represented in form of AST (Abstract Syntax Tree) and then tree matching corresponds to code matching techniques. We can make use of characteristic vector to represent the node of a tree as being done in DECKARD [3].

With **semantic based approach** we try to find to find the codes with more details and precision. In this the code is represented as graph, where nodes are statements and edges are dependencies (can be in form of function argument). They look for control flow and data dependencies both. We try to find similar isomorphic sub graph. Komondoor and Horwitz [9] used this technique and results were found out to be better from others.

IV. CODE DETECTION PROCESS

To understand these tools first of all let us discuss about some basic terminologies being used by most of these tools.
1. Code segment- This is nothing but a small part of the whole code. Usually developer is interested in this part of code only.
2. Code representation- This is the way a particular code segments will be represented by a tool so as to save them for their matching. Various ways are tree, tokens, graphs.
3. Code clone- This is a similar code segment to the one we are finding out. A code is said to be clone of other if they exhibit some form of similarity between them.

Various modules in a tool created for similar code detection are:

A. Code Repository
B. Pre-Processing
C. Transformation
D. Matching process
E. Filtering

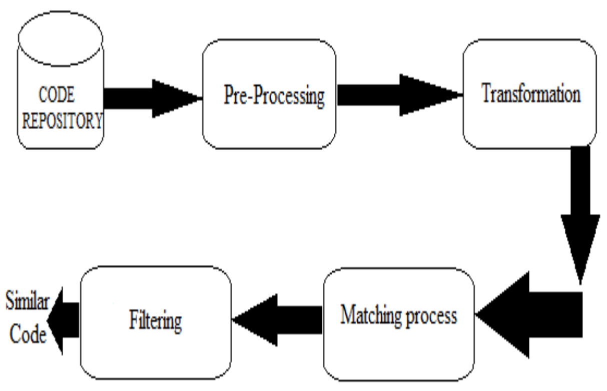

Fig. 1: General Process for finding code segments

The Figure shown above i.e. Fig. 1 is general overview of the techniques used by most of the tools. It is not necessary that all the tools follow this procedure. A main objective of all tools is to find code segments with great precision. Now we will discuss these stages one by one.

A. Code Repository

This is a database where all the programs made by different programmers are stored in some defined format so as to make easy work for tool to process them further. The code contained into this database has certain amount of validity. The code repository schema may vary from tool to tool.

B. Pre-Processing

This is first phase in which a tool will process the code so as to have only meaningful code. To have a meaningful code some part of the code has to be removed. For example, we need to remove SQL code embedded into JAVA code. After such removal, the code is partitioned into segments depending upon the granularity of the tool. For example, Code may be partitioned into lines, function units or tokens.

C. Transformation

After breaking the code into units, code is then transformed into forms similar to their matching technique used by tools. For example, AST in syntax based approach or Graph in Semantic based approach. During this transformation we need to eliminate white spaces in the code as they are done to have some of formatting in the base code. Also we can remove comments from the code as they are included only for programmer or user understanding. Inclusion of these things into detection process will only increase time and may affect efficiency of the tool.

D. Matching Process

After code being pre-processed and transformed they are fed into matching process. This is the heart of every tool and consists of an algorithm. Every tool have different matching algorithm. This phase compare the code of fixed granularity so as to find similar code. This process may also combine code granule to form the original code segment.

E. Filtering

This phase can be considered similar to ranking phase. The output of the matching process is feed into filtering process whose output is ranked order of the matching code.

This phase can be carried out manually which include intervention from human experts or automatically.

V. SCENARIO BASED EXAMPLE

Now let us see how the above diagram would be implemented in practice when working upon the code. This scenario is given in accordance to general schematic diagram Fig. 1 of the tools.

Consider following code:

```
Void addMult(int n){
   Float a=0.0f
   // Stores Sum of numbers
   Float m=1.0f
   // Stores Multiplication of numbers

For(int i=1;i<=n;i++){
            A=a+I;
            M=m*I;
}
return(a,m);
}
```

Consider the above mentioned code is stored in the code repository. This code is stored according to some predefined schema of the database. This schema is designed so as to make easy for tool to process.

In the pre-processing stage, the text based tool doesn't modify anything in the code; whereas in token based tool this code is broken into tokens such as addmult, int , float, for etc. This is the only stage were size of granule are decided for code.

Once the granularity is decided, we then proceed toward removing repeating tokens or removing white spaces or comments or punctuations marks etc. The code then would just be a sequence of token with decided granularity. No comments and punctuations will be there. After all this process, in syntax and semantic based tool we then go further by creating their respective structure. We create Abstract Syntax Tree (AST) in syntax based tool and Graph in Semantic based tool. In AST the body of FOR loop will be forming the sub tree of the node representing token of FOR loop. Also we can calculate characteristic vector if necessary for all the nodes. This process comes under transformation stage and is very specific to the tools.

Next stage is matching. In this stage based upon the granularity of the code comparison is made. In text and token based tool, comparison is made with simple text and tokens respectively. This type of matching is simple as compared to syntax and semantic based tools. Syntax and Semantic based tool matching are complex due to comparison between tree and sub trees and finding isomorphic graph respectively. Tree or graph matching algorithms are efficient but are bit time consuming.

The last stage is filtering. This phase can be done manually in which a team of expert decide the usefulness of found code and rank them accordingly. This technique is used by many tools. Other approach includes creation of a ranking algorithm which is used by DECKARD and many others.

VI. CONCLUSION

In this challenging environment lot of tools are available in the market for finding similar code segments or search code itself, as a developer these tools will alleviate him to find code as he is now able to look up similar code rapidly and efficiently. As from the point of view of developer their main concern is to reduce their effort and time, so these code search tools should consider this as their prime focus. These tools not only vary in their searching algorithm but also vary in terms of language they support. So a developer should choose a tool carefully.

REFERENCES

1. Collin McMillan, Mark Grechanik, Denys Poshyvanyk, Chen Fu, Qing Xie, "Exemplar: A Source Code Search Engine For Finding Highly Relevant Applications," IEEE, 2010.
2. Mu-Woong Lee, Seung-won Hwang, Sunghun Kim, "Integrating Code Search into the Development Session," IEEE 2011.
3. L. Jiang, G. Misherghi, Z. Su, and S. Glondu, "DECKARD: Scalable and accurate tree-based detection of code clones," ICSE, 2007.
4. J. Johnson, "Identifying Redundancy in Source Code Using Fingerprints," Proceedings of the 1993 Conference of the Centre for Advanced Studies on Collaborative Research, CASCON 1993, pp. 171–183.
5. U. Manber, "Finding Similar Files in a Large File System," Proceedings of the Winter 1994 Usenix Technical Conference, pp. 1-10.
6. S. Ducasse, M. Rieger and S. Demeyer, "A Language Independent Approach for Detecting Duplicated Code," Proceedings of the 15th International Conference on Software Maintenance, ICSM 1999, pp. 109-118.
7. B. Baker, "On Finding Duplication and Near-Duplication in Large Software Systems," Proceedings of the 2nd Working Conference on Reverse Engineering, WCRE 1995, pp. 86-95.
8. 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) pp. 654-670.
9. R. Komondoor and S. Horwitz, "Using Slicing to Identify Duplication in Source Code," Proceedings of the 8th International Symposium on Static Analysis, SAS 2001, pp. 40-56.
10. Simone Livieri, Yoshiki Higo, Makoto Matushita, Katsuro Inoue "Very-Large Scale Code Clone Analysis and Visualization of Open Source Programs Using Distributed CCFinder: D-CCFinder," in IEEE 2007.

AUTHORS PROFILE

Akshat Agrawal has completed graduation (B.E.) in Information Technology from University of Pune in 2011. Currently, he is pursuing her Master Degree (M.Tech) in Computer Science and Engineering from Lovely Professional University. He has keen interest in data mining and data warehousing. He has been researching in this field from his Bachelor degree and had made a project in the same area earlier.

Sumit Kumar Yadav has done his graduation (B.Tech) from UPTU in 2008 and received a M.S degree in Computer Science and Engineering from IIIT, Hyderabad, India. Currently, He is working as the Assistant Professor of Computer Science and Engineering in Lovely Professional University, Jalandhar. His current research includes Database, Data mining and Data warehouse. He has done a lot of research in this field and has published many paper in International Journals and Conferences.