POLARIS
live your dream

CODE CLONE DETECTION
A NEW APPROACH

- Sanjeev Chakraborty

CONTENTS

Sanjeev Chakraborty

Need of Research

Various literature surveys suggest that many programmers in the software development industry use copy-paste techniques, which are generally used as a form of reuse. Let's imagine our development environment where change request is implemented as a new function with minor changes in existing old function. Application may not use old function but the code is not removed. This situation can be compared with our Intellect Product-suite and various implementations of Intellect Product-suite in which few product functions are not used whereas new similar functions are written for client specific implementation using existing product code with minor modification. Both the functions or method (old and new) are present as a part of product for specific client implementation even if core product functions are not used.

This form of reuse usually results in code clones. Unfortunately, practicing copy and paste techniques can create very complicated maintenance tasks. In addition, among many other factors related to high memory utilization, code clones can create flaws in design due to poor cohesion between modules.

Assume working in maintenance based project giving frequent production issues. With initial evaluation, it is found that code changes need to propagate across many places, as design is distorted or modified. Identifying changes manually is difficult in case documentation is not available.

We all know that growing software becomes heavy day by day with age and high change requirement. The size of software increasing with growing age will lead to memory and performance related issues with existing infrastructure. This also raises the cost of maintenance.

All these issues are related to re-engineering of software, studied under ambit of reverse engineering, branch of software engineering. Reverse engineering is a broad research field, which starts from code and results to different artifacts that's gives information of design, and architecture of software.

Re-engineering of software can be done using software refactoring which includes various refactoring metrics and code clone detection. In this paper, we are discussing about code clone detection. The code clone detection (CCD) will detect the amount of clones available in project or application. This will also reveal the design related flaws or gap between design and implemented code. This will also state the amount of code can be refractor based on object oriented design flaws.

1. Abstract

The code clone can simply be defined as program fragment identical to another program fragment. The code fragment may be fully or partially identical. The definition states that when cardinality of intersection of two program entities exceeds a prescribed threshold, the two entities are considered as clones of each other. There is no prescribed threshold; each threshold is decided with some debate.

Available code clone detection techniques have their advantages and disadvantages related to quality of code clone and performance. This paper presents a new approach of code clone detection that does not depend upon threshold value given by the user for detection of a code clone; rather it detects all sets of clones with reasonable performance. The proposed approach detects block level clones using suffix trees that are created based on tokens. Tokens are generated using Abstract Syntax tree and clones are detected using existing type-1 and/or type-2 clone detection algorithms.

2. Introduction

Research shows that a large fraction of source code in many large-scale applications contains code clones. The existence of code clones can introduce much instability within a software application, unnecessary duplicates may create ambiguity. These instabilities can complicate routine maintenance tasks, since a change in one method may lead to changes across many methods. In addition, necessary duplicates can potentially induce the spread of bugs and prevent changes from being propagated. Therefore, in order to prevent and treat these instabilities, we must figure out where potential clones occur.

There are various techniques [1, 2, 7, 3, 8, 9, 10, 5] available for code clone detection including string based, token based, tree based, semantic based etc. Out of these token based [9] and abstract syntax tree based [2] approaches are more popular. Every technique has its advantages and disadvantages.

The approach of this paper is based on the concept of finding sub-clones and sequence clones for clone detection by Ira Baxter [2]. The approach that is described in this paper has two steps, firstly Abstract syntax tree (AST) generation and then suffix tree generation for clone detection. AST is used to detect statement clones. Further, block level clone detection is achieved through suffix tree using tokens that are generated from AST. This method does not require any threshold value to be suggested by user. This methodology depicts all sets of block level Type-1 and Type-2 [10] clones. Though the approach used in this paper is language independent, the tool that is developed based on it, is developed for Java. The remaining part of the paper is organized as follows. Section 2 discusses work done on code clone detection. Section 3 provides the new approach taken for code clone detection. Section 4 describes various steps involved in the new approach. The paper is concluded with the concluding remarks and future work.

3. Related Works

Various techniques can be found in the code clone literature to detect code clones [1, 2, 7, 3, 8, 9, 10, 5]. Along with the techniques there are number of tools available for various languages [7, 9, 2, 4, 6] Baxter et al. [2] proposed a technique of code clone detection using Abstract Syntax Trees (AST). In this AST based approach every sub-tree needs to be compared with every other sub-tree so the memory requirement is high. This also may hamper the performance and hence Baxter et al. used hash functions.

Kamiya et al. [9] have developed a code clone detector for various languages using token based approach. The paper discusses the process of identification of clones, issues that are faced while detecting the clones along-with the tool that is developed using the proposed technique.

Koschke et al. [10] discuss a technique that uses suffix trees to identify clones. In their technique first AST is generated which is serialized and then suffix trees are formulated. The technique helps detect type-1 as well as type-2 clones in an application.

4. Methodology

The methodology involves AST generation, token generation and suffix trees generation for code clone detection. ASTs are created for every statement.

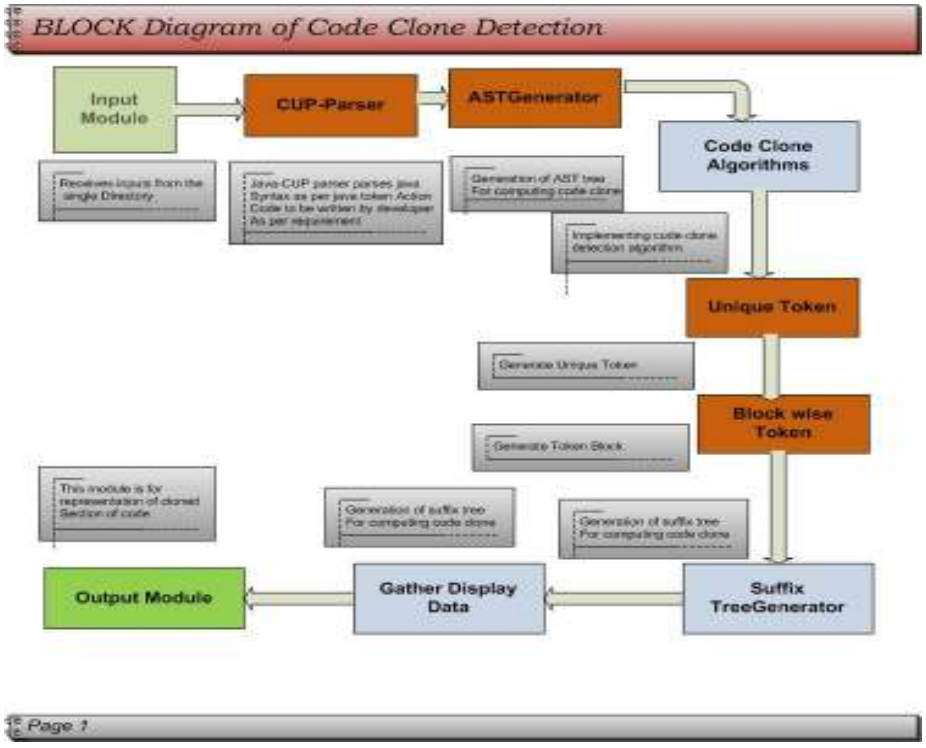


Fig 1:- Block Diagram of Code Clone Detection

A clone detection algorithm is applied to every sub trees of abstract syntax tree and unique tokens are assigned based on clone results. A suffix tree is generated with the token for block clone detection where every node of a suffix tree represents token with sequence of clone statement.

The code clone detection tool that is developed based on this approach currently covers only mathematical expressions in Java language. Further research work may scale up to other syntax of Java language. *Figure 1* presents the methodology used in this work.

5. Experimental Set-Up

A directory containing source .java files can be given as an input to the tool. Java Cup parser is used to parse the input. The action code is written in the parser to get a key value pair of position of syntax as a key and string as a value. The resulted string is then converted into the prefix format and a sub-tree of Abstract Syntax Trees is created for every statement. Code clone algorithms for Type-1 and Type-2 clones are then applied to these ASTs and Tokens are assigned based on clone results and stored in memory.

Only one or two lines of clones do not make true sense. Clone blocks can be detected using sequence of statements. Unique Token and further Block wise token generation are steps for sequence clone detection Note that if two ASTs are same then same token is given. Suffix tree is created using block wise tokens and clone sequences are identified. A single suffix tree is generated for detection of all clones. Traversal of the same branch of the tree helps identify the clones. This helps to reduce the memory requirement for storing and traversal effort is also less. Nodes of the suffix tree store information of the sequence clones information.

6. Implementation

In this section we discuss about the different modules used in implementation. Each module is explained with a brief description of the algorithm.

6.1 Input Parsing

Java Cup Parser is used in this code clone detection system. The Parser is used to parse .java files and desired output can be made available as per the action code implementation. The Action-Code written into parser would help us to prepare input as a key-value pair. Key would be position of syntax with structure as File No, Method No, Block No, Line No and value would be statement as a string object corresponding to the lone no. Figure 2 shows the arrangement of the data structure.

6.2 Abstract Syntax Tree (AST)

ASTs can be generated in different ways and formats. There is no standard specification for generation of AST OMG is working on standards of abstract syntax trees. ASTs generated by different compliers are different and depend on the requirements [6]. In this system, AST is generated with reference to the approach discussed by Ira Baxter [2].

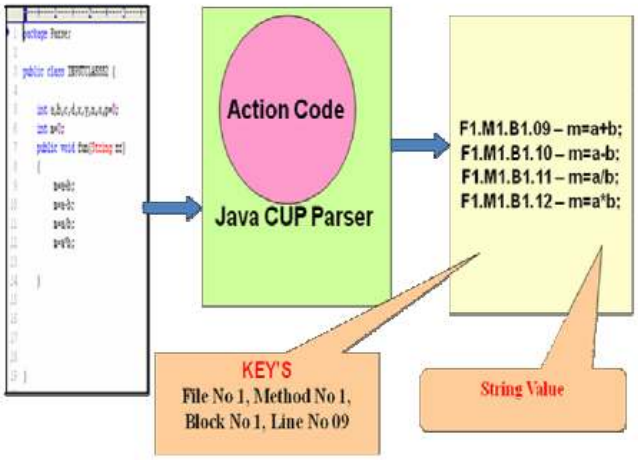


Fig 2 : Code parsing with key value

6.3 Type-1 and Type-2 Clone

Sub-trees are Type-1 clones if the entire tree including operand and operator of sub-tree is equal. Sub-trees are Type-2 clones operator node of both subs tree is same but operand may not be same.

For Type-2 clones, operator nodes must be same but operand nodes can be different. Sub-trees in this case are structurally same. The applied code clone algorithms gives result in Key Value format.

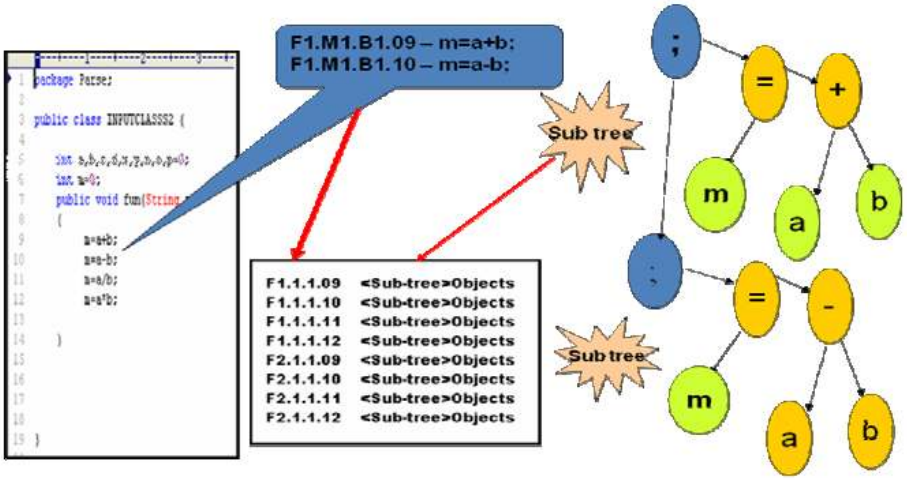


Fig 3: Clone detection using sub-Tree

6.4 Unique Token Generation

Every sub-tree is given a key as explained earlier. These keys are then read sequentially and a unique token is assigned to the complete sub-tree Keys that are same and are assigned same token. For example as shown in Figure 4, for Key 1.1.1.09 token is assigned as T1. For the next key next token is generated. Again next key 1.1.1.10 has token T2.

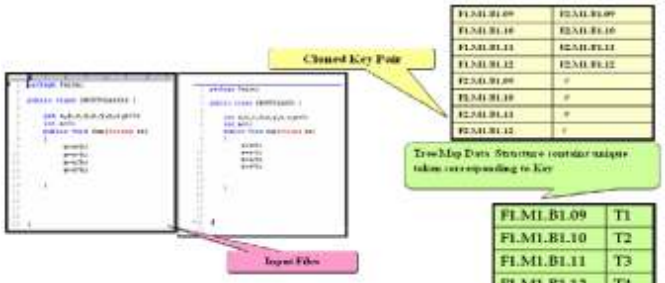


Fig 4 : Clone detection using sub-Tree

Token generation algorithm

- *Those key which are same will be assigned same token (say T1)*
- *Next token will be generated with adding 1 in subscripts (ieT2)*
- *On every iteration it will also check the for already assigned token against key*
- *In case the key are not matched, a new token will be assigned*.

6.5 Block wise Token Generation

This process is essential to detect the block level clone as tokens are divided into groups called block, which is a block of code starts with opening brasses and ends with closing brasses.

In this, compare all keys (e.g. 1.1.1.09) with each other except line no. Any difference found will lead to add a new Block Token with key value structure.

F1.M1.B1.09	T1
F1.M1.B1.10	T2
F1.M1.B1.11	T3
F1.M1.B1.12	T4
F2.M1.B1.09	T1
F2.M1.B1.10	T2
F2.M1.B1.11	T3
F2.M1.B1.12	T4

F1.M1.B1.09	T1
F1.M1.B1.10	T2
F1.M1.B1.11	T3
F1.M1.B1.12	T4

F2.M1.B1.09	T1
F2.M1.B1.10	T2
F2.M1.B1.11	T3
F2.M1.B1.12	T4

Fig 5 : Clone detection using sub-Tree

For example, as shown in figure 5 keys from 1.1.1.09 to 1.1.1.12 are in same block except line no and from 2.1.1.09 another block token is added. These block token are used to generate suffix tree.

Algorithm

- *Compare all key with other except line no*
- *In case any difference found, An another Tree Map will be created and key and token will be inserted till new mismatch arrives*
- *Before adding new Tree Map old Tree Map is added to Array List*
- *This continues till Tree Map ends*

6.6 Suffix Tree Creation

A single suffix tree is created using different block level token. One method is considered as a single largest block for which suffix tree is drawn. This helps to identify the largest as well as smallest clone block. Suffix trees are usually used to solve string problems that occur in text-editing, free-text search.

The longest repeated substring can be found in O (n) time using a suffix tree. Every node of suffix tree is a token and hence a statement. Largest branch of a suffix tree is equivalent to a method. Example of a suffix tree that is generated for the example discussed earlier is shown in Figure 6.

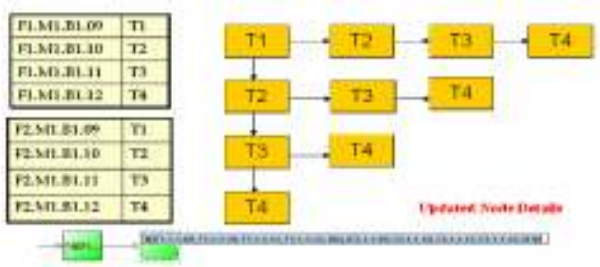


Fig: 6 Generation of Suffix tree

Suffix Tree Creation

- *Creation of suffix tree also updates clone key corresponding to token*
- *In case the token matches with current then also key information will be added or updated*

6.7 Probing Different Clone set

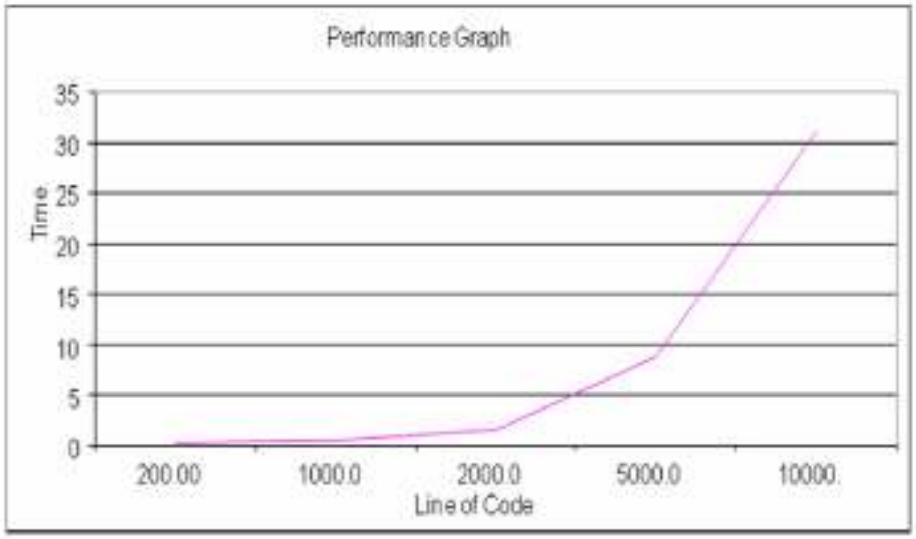
Probing different clone set will essentially segregate the different clones sets from the clone sub-set. Suffix tree is traversed and clone data is aggregated in an array list. This list contains all cone results. The result of clone sets are prepared by applying insertion sort with replacement or adding of node data. The clone sets, which are subsets of other sets, are replaced with super-set. The clone sets, which are not subsets of others, are added as new set.

[F1.M1.B1.09, F1.M1.B1.10, F1.M1.B1.11, F1.M1.B1.12]

[F2.M1.B1.09, F2.M1.B1.10, F2.M1.B1.11, F2.M1.B1.12]

Fig 7 Clone sets after probing

7. Testing and Performance

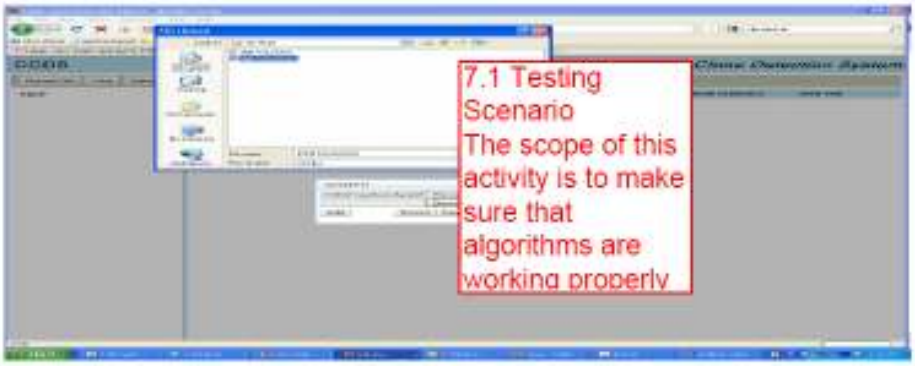


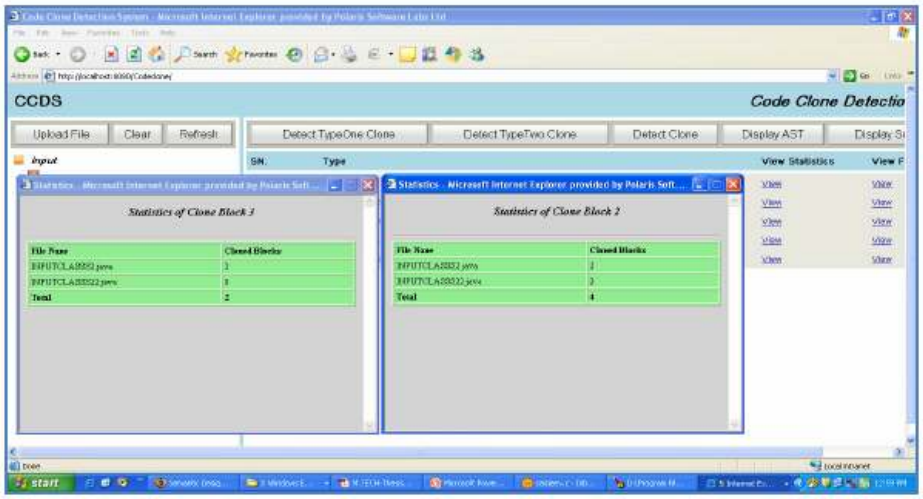
7.1 Testing Scenario

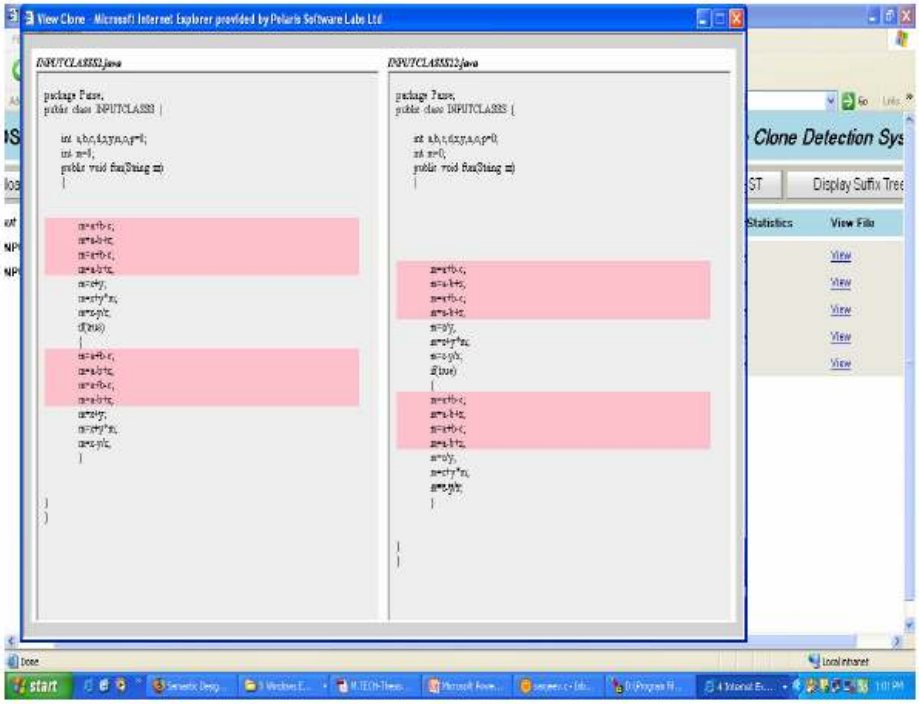
The scope of this activity is to make sure that algorithms are working properly with expected results.

- This testing shows the time required to compute clone detection.
- Graph prepared with Time in minutes and Total Line of Code.
- The testing has been carried out in development environment.

8. Walk Through of Code Clone Detection System




CCDS
Code Clone Detectio
Upload File
Clear
Refresh
Detect TypeOne Clone
Detect TypeTwo Clone
Detect Clone
Display AST
input
SN.
Type
View Statistics
Statistics of Clone Block 3
Statistics of Clone Block 2
File Name
Cloned Blocks
Total


View Clone - Microsoft Internet Explorer provided by Polaris Software Labs Ltd
INPUTCLASS2.java
INPUTCLASS22.java
Clone Detection Sys
Display Suffix Tree
Statistics
View File
View

9. Conclusion

This paper proposes methodology and architecture based on new approach of code clone detection that does not depend upon threshold value given by the user for detection of a code clone; rather it detects all sets of clones with reasonable performance. In this approach, initially sub-trees are created using abstract syntax tree with the source files given as input. Clone detection algorithm is applied on every sub tree and tokens are generated. Further Suffix tree is created based on block wise tokens generated and clone sequence are detected for blocks level clones of statements or expression with suffix tree.

Core implementation of CCDS (Code clone Detection system) lies on Clone detection algorithms, token generation algorithm, probing different clone set. The system has been tested with different test cases within scope of project. This system (prototype) is implementation of proposed methodology. Further, this prototype can be scaled up to product, which addresses clone of all java syntaxes by modifying AST generation modules.

10. Future work

The scope of this research was to implement the proposed methodology and validate the new approach with algorithms. The tool developed is a prototype with limited scope, but it can be further scaled up to a product.

- The project may be scaled to detect clone code of other types of java syntax.
- Input criteria may also be scaled to multiple directories with java source files.
- The functionality can also be extended with detection of TYPE-3 clones.
- The project can also be scaled up to detect clone of other languages with the change of parser and AST generation component.

11. References

[1] H. A. Basit, S. J. Puglisi, W. F. Smyth, A. Turpin, and S. Jarzabek. Efficient token based clone detection with flexible tokenization. In The 6th Joint Meeting on European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering: companion papers, pages 513–516, New York, NY, USA, 2007. ACM.

[2] I. D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier. Clone detection using abstract syntax trees. In Proceedings of the International Conference on Software Maintenance (ICSM), pages 368–379,Washington, DC, USA, 1998. IEEE Computer Society.

[3] B. B.S. On finding duplication and near-duplication in large software systems. In Proceedings of Second Working Conference on Reverse Engineering (WCRE), pages 86–95, Toronto, Ontario, Canada, July 1995. IEEE Computer Society

[4] E.Duala-Ekoko and M. P. Robillard. Clone tracker: Tool support for code clone management. In Proceedings of the 30th international conference on Software engineering (ICSE), pages 843–846, Leipzig, Germany, 2008. ACM.

[5] K. Greenan. Method-level code clone detection on transformed abstract syntax trees using sequence matching algorithms. Technical report, 2005.

[6] http://clonedigger.sourceforge.net/. Clone digger.

[7] Y. Jia, D. Binkley, M. Harman, J. Krinke, and M. Matsushita. Kclone: A proposed approach to fast precise code clone detection. In In IWSE, 2009.

[8] L. Jiang, G. Misherghi, Z. Su, and S. Glondu. Deckard: Scalable and accurate tree-based detection of code clones. In Proceedings of the 29th international conference on Software Engineering (ICSE), pages 96–105, Washington, DC, USA, 2007. IEEE Computer Society.

[9] T. Kamiya, S. Kusumoto, and K. Inoune. Ccfinder:amultilinguistic token-based code clone detection system for large scale source code. IEEE ransactions On Software Engineering, 28(7):654–670, July 2002.

[10] R. Koschke, R. Falke, and P. Frenzel. Clone detection using abstract syntax suffix trees. In Proceedings of the 13th Working Conference on Reverse Engineering, pages 253–262, Washington, DC, USA, 2006. IEEE Computer Society.

About Polaris

Polaris Software is the world's most sophisticated banking and insurance software company. Over 1,000 engineers across four R&D centre's, work on Service Oriented Architecture (SOA), enabling non-disruptive modernization at the lowest total cost of ownership. Over the last two decades, Polaris has implemented its solutions and services across 200 of the world's largest financial institutions. Polaris is acknowledged by Banking and Technology Market Analysts as a leading provider of BFSI Technology Solutions. Polaris provides mission critical services to some of the largest financial institutions in the world. With 10,000+ team members. And a physical presence across 29 countries around the world.

About Author

Mr. Sanjeev Chakraborty is part of Enterprise Architecture, CTOO Team, involved in architecture design and security consultancy to various project implementation in finance. Responsible for architecture and security design and assessment, web vulnerability solution, application of PKI, cryptography, authentication and authorization with industry standard security practices. He has participated in various conferences and workshop related to ICT, application security, parallel computing and software engineering organized by DIT GOI. IBM, IIT Madras and SIGSE of Computer Society of India. His area of interest is architecture design, security solution along with applied research in software engineering, application security and reverse engineering

Corporate Headquarters:
Polaris Software Lab Limited
'Foundation', # 34, Rajiv Gandhi Salai,
Chennai - 603103. INDIA.
Phone: 91-44-27435001 / 91-44-39873000

• Bangalore • Belfast • Chennai • Chicago • Dubai
• Dublin • Frankfurt • Hong Kong • Ho Chi Minh Cit
• Hyderabad • Illinois • Kuala Lumpur • London
• Madrid • Manama • Melbourne • Mississauga
• Mumbai • Neuchatel • New Delhi • New Jersey
• Paris • Pittsburgh • Pune • Riyadh • Santiago
• San Francisco • Seoul • Shanghai • Singapore
• St. Germain En Laye • Sydney • Thane • Tokyo
• Toronto • Utrecht • Victoria • Wicklow

www.polarisFT.com