Journal of Information Engineering and Applications
ISSN 2224-5782 (print) ISSN 2225-0506 (online)
Vol.5, No.2, 2015

Three Levels Analytical Model for Monolithic Legacy Program Source Code Analysis

Asfa Praveen[1*] Shamimul Qamar[2] Shahanawaj Ahamad[3]
1.Ph.D. (Computer Science) Research Scholar, Faculty of Science & Technology, Shri Venkateshwara University, Gajraula, (U.P.), India
2.Professor of Electronics & Computer Engineering, Noida Institute of Engineering & Technology, Greater Noida, (U.P.), India
3.Assistant Professor, Dept. of Computer Sc. & Software Engg., College of Computer Sc. & Engg., University of Ha'il, Saudi Arabia
* E-mail of the corresponding author: asfa_praveen@yahoo.com

Abstract
Source code analysis has many different but related and connected perspectives. As per the requirement of each perspective, the program code has been analyzed by specific technique. No single technique is able to achieve the purpose and covers all aspects of analysis. It addresses issues that are coupled together to entail development concerns, that will directly or indirectly affect the functionality of the program. They are categorized with reference to their application in the program analysis process. This paper describes all those analysis techniques organized in three levels analytical model which are applied to analyze and enquire the source code and its different aspects.
Keywords: Monolithic, Source code, Program analysis, Walkthrough, Syntax analysis.

1. Introduction
This paper describes the details of proposed approach named as three levels analytical model. This model is based on application of various analysis techniques which are used to analyze supplied monolithic legacy source code according to specific perspective. Each analysis technique applied at level-I produce the results. These results are managed to store with creating a central a knowledge base by general database technique in level-II. This knowledge base contains much useful information about the program. The central knowledge base further used to generate the reports at level-III. These reports are about program, syntax, semantic, dataflow, code clones etc. The generated reports will play a crucial role for further development of monolithic legacy code.

2. The Three Levels Analytical Model
All The technological concerns and the business nature of the program will always lead to the process of the analyzing the program legacies. It is aimed at avoiding some risky processes of making need of unnecessary corrections on the code. This may also make the functionality of the code be tampered with. The model is depicted in figure 1. The details of each analysis technique used in proposed three levels analytical model are explained after the figure.

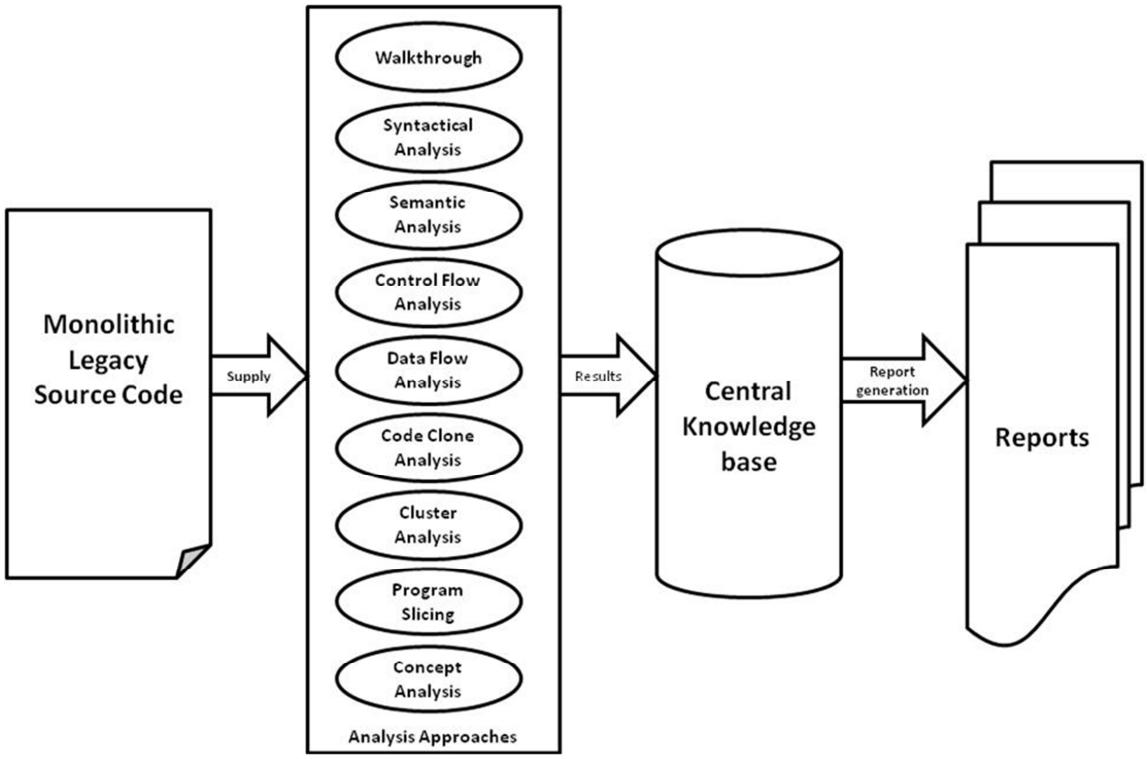

Figure 1.Three levels analytical model

2.1. Walkthrough

First, Walkthrough (OWASP 2008) is a thorough technical review process of code. It includes the review of code structure (Weber 1997). The line by line study is performed to see what is code functioning (Woody III 2010). It also analyses, for what purpose each logical code unit has been implemented. It is beneficial for many aspects as to find out the followings:

i. Features of code;
ii. Objects of program;
iii. Integration of modules;
iv. Higher level abstraction;
v. Modules hierarchies;
vi. Data flow management;
vii. Volume of data;
viii. Code defects;
ix. Logical flow;
x. Data dictionary.

Success of walkthrough depends on the skills of the reviewers, their experience with same domain of programming language and the developed code's artifacts. Because of this reason it is considered as cumbersome and time consuming activity. Once the code walkthrough has been performed, it can bring a clear picture of functionalities, data and logics. Walkthrough documents (SWPG 2011) should be prepared with writing all the details of review process and findings, which will be helpful for further evolution process and target system verification.

2.2. Syntactical Analysis

The syntactical analysis (Hilda 2002) is performed to get the information about validity of series of tokens. Reserve words, identifiers and special symbols are called tokens. It is not necessary that the sequence or series of tokens may be meaningful; it checks its validity that is there anything wrong as per the syntax statement composition rules. For example "False-2" is valid but not have some sense, its sense will be identified and analyze in semantic analysis phase of this work.

The syntactic analysis is performed with the help of parsing and context free grammar techniques (Floyd 1964). The parser get a token from the lexical analysis phase (Shinomi 2010) and produce an abstract syntax tree (AST) or parse tree structure. It is used to represent the structure and composition of program statement. Figure 2 show is the process of formation of parse tree from tokens.

It is a process of analyzing the entire program by use of the symbols that were used in coming up application. It involves the use of various parsing techniques in its structure when it comes to analyzing the code and the application in general. The tenacity behind syntactic analysis is to focus the structure of the input content. This structure comprises of a progression of expressions, the lowest of which are the essential token and the biggest of which is the sentence (Trivedi 2013). The can be depicted by a tree with one node for each expression. The base of the tree denotes the sentence. They help in analyzing the error level in the program and try to debug in order to get the best deliverable results.

Calculations over the information can be composed with property linguistic use determinations that are focused around a dynamic grammar. The conceptual sentence structure portrays the structure of a theoretical punctuation tree, much the way the solid grammar depicts the expression structure of the data. For a proper functionality of the program, it would be vital for the two concepts discussed be taken into consideration. This concept will be able to detect the occurrence of errors within some lines if codes. Proper punctuation of the code segment is also missing at some points that will return more error messages. The concept will help in eliminating the ambiguity in the program. It will ensure that all the symbols used are understandable to the maintainer.

Top-down, bottom-up and centered lexical analyses are main class of algorithm for parsing (Eastman 1983). These are the conceptual procedures which entails the analysis of the symbols used in the code segment. It can be in either language, natural language or even in the computer languages. It will take into use the normal rules and formal grammars that are used in constructing the program. Languages could be believed to be a probable combination of syntax analysis, which is the much concerned with the analysis of the context free grammars approach in the collection of codes.

It majorly entails the analysis of the grammar used in building an application. It applies the concept of parsing. This is the concept of making some of the programs code segments be understood by the analyst. It implies part of analysis task. It gives the interpretation of entire program and grammar of used in coming up application.

The concept of parsing in the program analysis takes some major approaches. This includes Top-Down approach of analyzing the code. This may include some other sub sections like the case of the syntax analysis. This is a recursive descent mechanism.

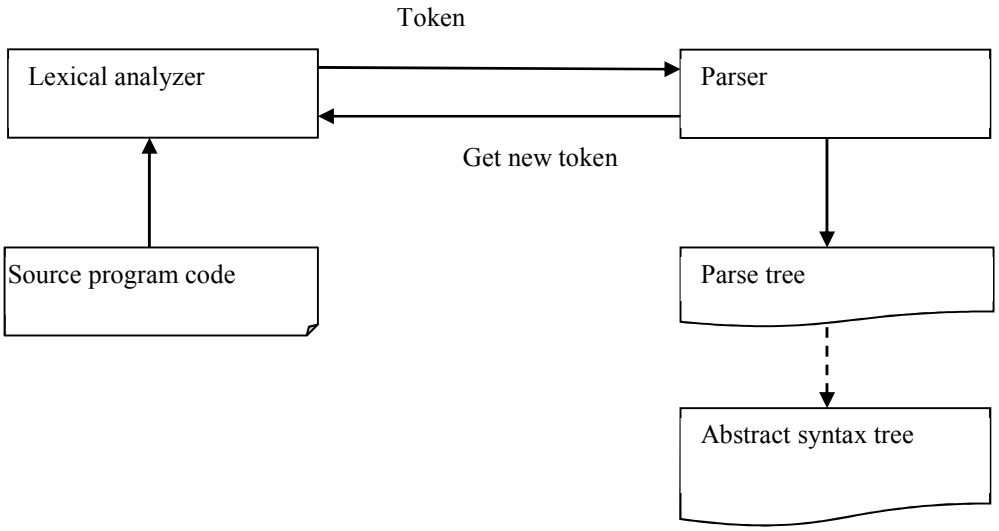

Figure 2.Parse tree formation from tokens

2.2.1. Top-Down Analysis Approach: Top-down analysis of program is the inverse of a base up program analysis. It refers to a style of analysis where an application is analyzed from the beginning with an abnormal state representation of what it should do. It is separating the particular into easier and less difficult pieces, until a level has been arrived at that relates to the primitives of the programs.

Top-down analysis has a tendency to analyze modules that are focused around usefulness. In this style of analysis, there is an incredible risk that execution subtle elements of numerous information structures must be imparted in the middle of code modules, and consequently all around uncovered. This thusly makes it enticing for different code modules to utilize these execution points of interest, accordingly making undesirable conditions between diverse parts of the application's code.

2.2.2. Bottom-Up Analysis Approach: It is another approach which is used in the analysis of legacy code. It at some points plays the same role as that of the top down approach. It is only the opposite approach into coming

up with a solution from top-down. It consists of analysis of various artifacts as data type, text, numbers and main program components. It also focuses on to find a relationship among all these. This analysis technique (Paul 2007) has been very popular in recent years not only in complex code even in other sectors as testing, comprehension, data mining etc.

2.2.3 Lexical Analyzer: Lexical analyzer is a tool that performs analysis which includes the extraction of individual words, lexemes from program code. Different parts of the lexical analyzer incorporate the evacuation of whitespace and remarks and taking care of compiler mandates. The tokenization methodology takes data and afterward passes this info through an essential word recognizer, an identifier recognizer, a numeric consistent recognizer and a string steady recognizer, each one being placed into their yield focused around disambiguating tenets (Shen 1980).

The program under focus is passed to the compiler style lexical analyzer and broken into tokens. The resulting token sequence should be then scanned to identify duplicated tokens while the original source code is kept as clones. The token then should be non parameter and parameter tokens by use of the lexical analyzer. Parameter tokens are the identifiers and literals in the source code. The lexical analyzer further separates the non parameter token found on a line by a hash factor and encodes the parameter tokens by a position index. The lexical examination methodology begins with a meaning of what it intends to be a token in the dialect with normal articulations or language structures, then this is meant an unique computational model for perceiving tokens, which is then meant an implementable model for perceiving the characterized tokens.

2.3. Semantic Analysis

Syntactical analysis generally verifies that the tokens are arranged in language grammatical correct order nothing is wrong as per the rules of language grammar context. After performing the syntactical analysis (Landauer 1998) of legacy code, it is very useful to analyze its semantics. It investigates whether the correct or valid language tokens arranged and written in a meaningful way or have some meaningful sense (Allen 2009). The semantic information can be sourced from the identifier names for the source code or the comments and documentation exist. Semantic analysis provides the legacy program with tools as well as services that help to service orient the program. To carry out a semantic analysis for the program under focus, a static program should be used to provide information on the semantics of the source code. It represents the monolithic legacy program in a program dependency graph (PDG). Once the program is represented as a graph, the nodes represent the statements and the expressions. On the other hand, the dependent data and controls are represented relatively. Therefore, it is easier to understand the source code. Semantic analysis generally includes the tracking of following according to their correct, valid, meaningful occurrence and implication to a correct result.

i. Variable declaration and scope;
ii. Function declaration and call;
iii. Types and declarations;
iv. Static and dynamic type checking;
v. Identifiers and its usability;
vi. Type conversions;
vii. Constants;
viii. Expressions;
ix. Arrays;
x. Struct and enums;
xi. Pointers;
xii. Checking begin and end of code modules.

Above mentioned are given as example elements for investigation but not limited to. This analysis can include many more code elements to check the meaningfulness of artifacts found in legacy code.

2.4. Control Flow Analysis

Control flow analysis is an analysis technique that analyzes the control flow in a program (Nielson & Hankin 2011). It is analysis of sequence of operations. Basically the control flow of a program is depicted by flow chart but at higher level a control flow graph (CFG) is prepared to show the control flow (Midtgaard 20YY). To produce CFG following elements must be analyzed.

i. Iterative statements;
ii. Branching instructions;
iii. Jumping instructions;
iv. Use of unconditional jumping as “goto”;
v. Dominators;
vi. Dependence analysis.

The figure 3 presents a code segment and figure 4 presents its control flow graph.

Line No.	*Code*
1	x := 0
2	y := x * y
3	Lebal1: z := y/d
4	if c < y goto Label2
5	p := y / z
6	f := p + 1
7	Label2: g := f
8	q := i - k
9	if p > 0 goto Label3
10	goto Label1
11	Label3: return

Figure 3.Sample code segment with unconditional branching

CFG of above code segment is as follows:

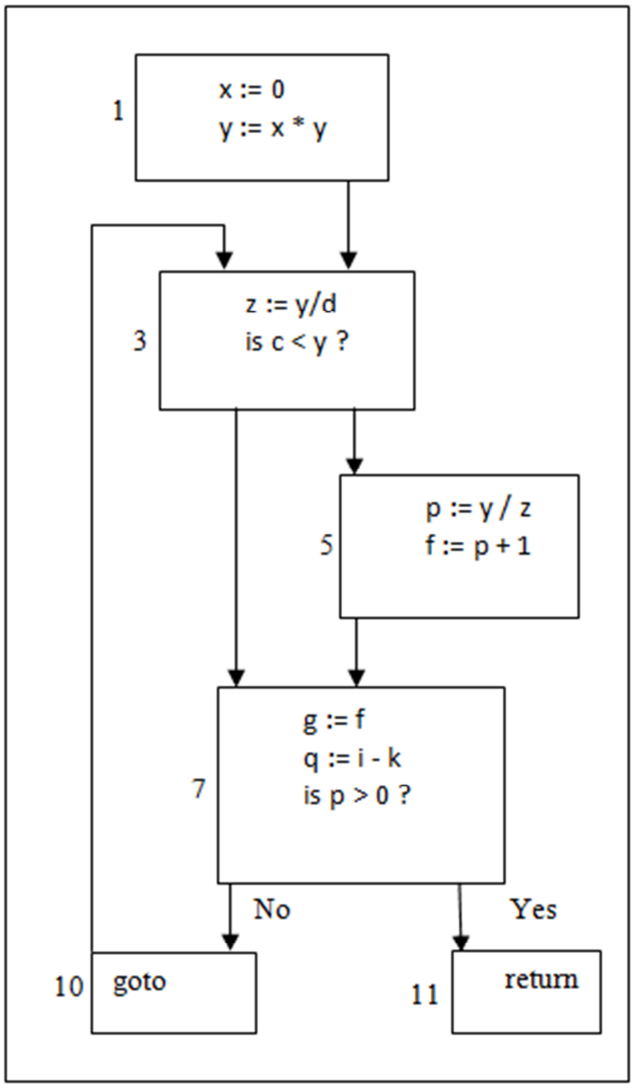

Figure 4.Control flow graph

2.5. Data Flow Analysis

Data flow analysis (Foster 2011) is conducted to obtain information for finding the values of variables or data holding entities at various stages in program. Control flow graph is used to access those stages in program where a value is assigned to variable and the stages up to which it is flowing. The blocks of control flow graph are

represented as stage of program. Following should perform to conduct data flow analysis.

i. Derive data flow equations for each node of the control flow graph;
ii. Information at the boundaries of blocks;
iii. Exit state of a block;
iv. Use of iterative algorithm for data flow equations;
v. In state and out state of blocks;
vi. Forward analysis and reaching definition;
vii. Backward analysis;
viii. Live variable analysis;
ix. Removal of dead code;
x. Document data flow.

2.6. Code Clone Analysis

The process of clone detection recognizes repetitive bits of source code created by copy and paste program development technique. About whether, these redundancies are liable to be altered conflictingly and, consequently, bring about blunders. Further, excess source code blows up the estimation unnecessarily and builds support costs. The code clone is a duplicate code written somewhere in program (Prem 2013). The need to incorporate the code clone is for easy program development when some similar or near to functionality is required. Already written code is copied and pasted at other part as it is or slightly modified, it creates easiness while developing but create a big problem while maintaining the program. It has become very difficult to identify that which clone should keep or which has to be removed. On the other hand, while updating one code then it is essential to update the corresponding clone (Miryung 2005). It produced the side effect too. Considerably more discriminating is the risk of presenting bugs, the designers who change a bit of code are generally not mindful that duplicates exist inside the module. Therefore, inconsistencies can emerge which can result in bugs. Studies demonstrate that around half of all unintentionally presented changes are really mistakes.
Identification of clones can be used by analyzing the graphs developed during the syntax or semantic analysis. To identify the code clone for the legacy program under focus recommend the use of the parse tree developed after the syntax analysis. The tree matching approach is used to find similar sub trees hence finding clones. If the names of the variables, tokens and literal values have been abstracted; sophisticated methods are required to find the clones. For the code segment under focus, use dynamic programming technique. The code is divided in blocks and the blocks are compared to each other by analyzing each statement of a block against another statement of a block. The minimum distance between the clones is used in the hypothesis where by the smaller the minimum distance, the higher the likelihood of the existence of the clones. However, some clones are difficult to identify. Therefore, the clones can be identified using trained neural networks. The neural networks are trained to identify the clone according to the features that they have been given of the hard to get clones.

Teamscale (cqse 2014) utilizes a very powerful repetition examination innovation that has been exhibited in numerous experimental commitments on universal gatherings. Teamscale is not just equipped for recognizing accurate duplicates of code sections yet utilizes standardized procedures and can in this manner even report clones that contrast in some unessential parts. By utilizing replicated code redundancies are brought into the code-base that raise viability endeavors on the grounds that adjustments must be rehash in different spots. CCFinder (ccfinder 2014) and CloneWarrior (Khan el.al. 2014) are helpful tools to identify the code clone.

2.7. Cluster Analysis

Cluster analysis technique is used to look for interrelated items in a data-set. For the monolithic COBOL legacy analysis of the variables leads to establishment of a set. Variables are analyzed by looking at their distribution in the legacy code. The variables should be separated between important and unimportant variables using data persistence and manual inspection by the person performing the legacy program analysis. Then the location of a variable in the code is determined as well as its usage per section of program. The results should be placed in a matrix that is used for the analysis. The hamming distance, which is the distance between two variables, is calculated by subtracting the matrix obtained for one variable form the matrix of another variable. The longer the distance, the dissimilar the matrix and the variables can be. The dissimilarity can be used as an input in the clustering algorithm. To analyze the clusters of the COBOL program use agglomerative hierarchical clustering algorithm (AGNES). The AGNES algorithm places every variable of the program in its particular cluster. The algorithm then creates new clusters that hold two or more neighboring clusters. The algorithm will terminate if only one cluster remains.

2.8. Program Slicing

Slicing is the abstraction of the monolithic legacy program based on the behavior. A slice is an executable part of the program. Slicing helps in breaking the program into small parts that can be easily analyzed by the

maintainers. It also allows analysis of the program according to the data flow. Therefore, maintenance engineer is able to understand the working of the program. To slice the program on focus it uses a slicing criteria <P, V>, where V is the variable at a given point P in the program. The slice is made up of all program statements that may have an effect on the value V at the point p.

To slice the legacy program to tokens, use a work on the dependence graph. A work on the dependence graph is a graph that shows the dependency of objects to one another. Develop the graph for the legacy program and then insert the start node and the end node. Then to slice the program, traverse the graph from a given point P in the graph tracing back all nodes that directly or indirectly depend on the node. Again using the data flow analysis, trace all the nodes from the point P that have a dependency on the graph according to the data flow. Mark all the visited nodes and delete the unvisited nodes. The remaining graph makes a token of the program. The token is used to analyze the legacy program. Other tokens can be realized by choosing different points P and then tracing the dependent nodes.

However, slicing the program in this manner is challenging due to the intensity if that data flow analysis. To overcome this challenge, it is recommended to use the creative approach that incrementally analyses the data flows. Once an analysis of the data flow is done, there is no need to analyze it again in the next point of slicing the program.

On the other hand, it also refers to the computational technique of a given combination of statements that are believed to be affecting the program at some given point of the execution process (mharman 2014). It is vital in the debugging process to help in realizing the source of errors that could exist in a program or just within the segment of a code. Static and dynamic slicing (Agrawal 1990) are two forms of the techniques. It involves the original statement that is believed to be affecting the program operation. It tends to reduce the cases of the redundancy in the program segment (Weiser 1984).Like, for example, in the code segment below, written in C language;

```
int i;
int sum = 0;
int product=1;
for (i=1;i<N; ++i) {
sum=sum+i;
product=product*i}
write(sum);
write (product);
```

Figure 5.A sample C code

A sliced component of the program is given by;

```
int i;
int sum = 0;
for (i=1;i<N; ++i) {
sum=sum+i;     }
write(sum);
```

Figure 6.A sample sliced code segment

2.9. Concept Analysis

Concept analysis of the legacy program is closely followed by cluster analysis. The techniques are mostly used to detect objects within the source code. Concept analysis is a method for analyzing the relationship between the entities and their variables (Ganter et.al. 2003). Conduct a concept analysis for the identification of entities and their variables is the first step. For instance in a K = (R, O, A) binary equation R and O represents the set of objects and attributes respectively while A is the binary relationship. The concept in this case is represented by the set sharing the common attributes. The use of the Birkhoff's concept lattice (Ganter et.al. 1980) to find all the concepts for the program is recommended.

Using the deterministic approach to traverse the graph, a node should be labeled to have a certain attribute if it is the most general attribute in the concept. On the other hand, a labeling of an object to a node, the most specific object of the concept is selected. Consequently, it is simple using the graph to identify the objects, their attributes and their relationships. To identify the attributes for a given node, get all the attributes on the node as well as the attributes above the node. On the other hand, the objects of the node are objects on the node and those below the node. An attribute therefore is used on the same node and below it.

3. Conclusion & Future Work

Program analysis has been a critical activity in all type of development of legacy programs. The work presented

in this paper proposes a model that is consisting of many sub-activities for analysis of program source code to investigate and understand according many aspects of program and its source code. The study is based on monolithic type of program code. Model has three levels. This paper has explained the procedure adopted to perform activities in first main analytical level. Rest of the two levels have undertaken for further research work. The future research will explain that how the derived and extracted data, information will be stored in central knowledge base and in what form? What will be methods and procedures for retrieval of information? The retrieved information will be generated in the form of reports in the last level of proposed future work. The research work is able to analyze the monolithic type of program source code significantly and successfully.

4. References

OWASP (2008), Code Review Guide V1.1.
Weber , S. (1997), "Code Walkthrough Guidelines", Cornell University.
Woody III , L. S., (2010), "Code Review Procedure".
SWPG (2011), Structured Walkthrough Process Guide, Commonwealth of Pennsylvania, Department of Public Welfare.
Hilda K., Dominique, S. & Edward, S.(2002), "An Introduction to Syntactic Analysis and Theory".
Floyd, R.W. (1964), "The Syntax of Programming Languages-A Survey", Computer Associates, Inc.
Shinomi, H. & Ichimori, Y. (2010), "Program Analysis Environment for Writing COBOL Aspects", AOSD '10, ACM.
Trivedi, A. & Ugrasen, S. (2013), "Design of a Reverse Engineering Model (A Case Study of COBOL to Java Migration)", International Journal of Computer Applications, 79(5), 0975 – 8887.
Eastman, C. M. (1983), "A lexical analysis of keywords in high level programming language", Int. J. Man-Machine Studies, 19, 595-607.
Paul, K. (2007), "Quick Introduction to Syntax Analysis".
Shen,V. Y. & Dunsmore, H.E.(1980) "A Software Science Analysis of COBOL Programs",Computer Science Technical Reports. http://docs.lib.purdue.edu/cstech/278, 278.
Landauer, T.K , Foltz, P.W. & Laham, D. (1998), "An Introduction to Latent Semantic Analysis, Discourse Processes", 25, 259-284.
Allen J.F., Swift, M. & de Beaumont, W.(2009), "Deep Semantic Analysis of Text", University of Rochester.
Nielson & Hankin,(2011), "Control Flow Analysis", Harvard University.
Midtgaard, J. (20YY), "Control-flow analysis of functional programs", ACM Computing Surveys, V, 1-38.
Foster, J. (2011), "Dataflow analysis", Harvard University.
Prem, P. (2013), "A Review on Code Clone Analysis and Code Clone Detection", International Journal of Engineering and Innovative Technology (IJEIT), 2(12).
Miryung, K. & Murphy, G.C. (2005), "An Empirical Study of Code Clone Genealogies", ESEC-FSE'05, September 5–9.
cqse (2014), https://www.cqse.eu/en/products/teamscale/overview/
ccfinder (2014), http://www.abelssoft.net/ccfinder.php
Khan, K., Saif Ur Rehman & Fong, S. (2014), "A Comprehensive Study of Finding Copy and Paste Clones from Program Source Codes", Journal of Emerging Technologies in Web Intelligence, 6(1).
mharman, (2014), http://www0.cs.ucl.ac.uk/staff/mharman/sf.html
Agrawal, H. & Horgan, J.R..(1990), "Dynamic Program Slicing", Proceedings of the ACM SIGPLAN'90.
Weiser, M. (1984), "Program Slicing", IEEE Transactions on Software Engineering, 10(4).
Ganter, B. et. al. (2003)," Formal Concept Analysis: Methods and Applications in Computer Science"
Ganter, B. & Wille, R.(1980), "Applied Lattice Theory Formal Concept Analysis".

Authors' biographies

Asfa Praveen has seven years of experience with good practical, academic and research projects exposures after completion of Master of Computer Applications (M.C.A.) in year 2007 from Punjab Technical University, Jalandhar, India with excellent grade; Advanced 'A' level (P.G.) Diploma in Computer Science in year 2003 from Department of Electronics, Ministry of I.T., Govt. of India; Oracle Certified Professional (O.C.P.) Examination in year 2003 from Oracle Corporation, U.S.A.; completing Ph.D. in Computer Science in Faculty of Science & Technology of Shri Venkateshwara University, Gajraula, (U.P.) INDIA. Her area of research includes Service Oriented Migration & Development of Monolithic Legacy Software.

Shamimul Qamar is working as full professor of Electronics & Computer Engineering. He has eighteen years of wide experience in research, academics and administration, held various positions as Director, Professor, Consultants in universities and engineering colleges after completion of Ph.D. in Electronics and Computer Engineering from Indian Institute of Technology (I.I.T.) Roorkee, India with excellent grade. He has completed

B.Sc. from Ch. Charan Singh University, Meerut; Bachelor of Engineering (B.E.) in Electronics & Communication Engg. from Madan Mohan Malviya Engineering College, Gorakhpur; M.Tech. (Information & Communication Systems) from Aligarh Muslim University, Aligarh. He has published more than 35 research papers in his credits and supervised many master projects and Ph.D. thesis.

Shahanawaj Ahamad is an active academician and researcher in the field of Computer Science, Software Reverse Engineering with twelve years of research and academic experience including six years in abroad, working with College of Computer Science & Engineering of University of Ha'il, K.S.A as Assistant Professor. Before joining UoH he has worked with King Saud University, Al-Khraj University of K.S.A. and Shobhit University, Meerut (Delhi-NCR) and Uttar Pradesh Technical University of INDIA as HoD-I.T., Asstt. Professor etc. He is the professional member of British Computer Society, U.K., senior member of Computer Society of India including membership of various national and international academic and research organizations, member of research journal editorial board and reviewer. He is currently working on Service-Oriented Migration, Multi Agent System Reverse Engineering, published more than twenty five research articles in his credit in national and international journals and conference proceedings. He holds M.Tech. followed by Ph.D. in Computer Science with specialization in Software Engineering from Jamia Millia Islamia Central University, New Delhi, India. He has supervised many bachelor projects, master and Ph.D. dissertations.

The IISTE is a pioneer in the Open-Access hosting service and academic event management. The aim of the firm is Accelerating Global Knowledge Sharing.

More information about the firm can be found on the homepage:
http://www.iiste.org

CALL FOR JOURNAL PAPERS

There are more than 30 peer-reviewed academic journals hosted under the hosting platform.

Prospective authors of journals can find the submission instruction on the following page: http://www.iiste.org/journals/ All the journals articles are available online to the readers all over the world without financial, legal, or technical barriers other than those inseparable from gaining access to the internet itself. Paper version of the journals is also available upon request of readers and authors.

MORE RESOURCES

Book publication information: http://www.iiste.org/book/

Academic conference: http://www.iiste.org/conference/upcoming-conferences-call-for-paper/

IISTE Knowledge Sharing Partners

EBSCO, Index Copernicus, Ulrich's Periodicals Directory, JournalTOCS, PKP Open Archives Harvester, Bielefeld Academic Search Engine, Elektronische Zeitschriftenbibliothek EZB, Open J-Gate, OCLC WorldCat, Universe Digtial Library , NewJour, Google Scholar