International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)
Web Site: www.ijettcs.org Email: editor@ijettcs.org, editorijettcs@gmail.com
Volume 2, Issue 3, May – June 2013 **ISSN 2278-6856**

Fault Localization for Dynamic Web Application

G.Maheswari[1], Dr.V.Venkatesakumar[2]

[1&2] Computer Science and Engineering, Anna University,
Regional Centre, Coimbatore-47, India

Abstract: ***Fault localization is the problem of formative where in the source code modifies has to be completed in order to fix the sensed failures. The cause of the failure is called as fault that also called as bug. Based on the dynamic execution of the web application segregate the source cause of fault by fault localization techniques. To localize faults efficiently in web applications, some fault-localization algorithms can be improved by using a comprehensive domain for conditional and function-call statements and using a source mapping but did not focus on cross based language in web application. This study use traverse analysis based technique for fault localization. A traverse analysis is constructed for each executable statement and to determine the suspiciousness of the corresponding statement. In addition, to use dynamic test generation based on heuristic search strategies. This approach directed by the control flow graph of the program beneath test.***

Keywords: fault localization, traverse analysis, web application, statistical debugging, test generation

1. INTRODUCTION

An erroneous step, process, or data definition in a computer program which causes the program to do in an unexpected behaviour. It is an intrinsic weakness of the design or implementation which might result in a failure. In program debugging, fault localization is one of the priciest processes. It can be further divided into two main parts. The initial part is to recognize apprehensive code by use a technique. It may contain program bugs. The next part is for programmers to actually examine the identified code to decide whether it certainly contains bugs.

In first part apprehensive code is prioritized based on its probability of containing bugs for reference all the fault localization techniques. Code have a lower priority must be examined after code have a higher priority, as the former is more suspicious than the latter. The next part, we assume that bug detection is perfect. So the programmers can forever correctly classify faulty code as faulty and non-faulty code like non-faulty. The amount of code desires to be inspected may increase when perfect bug discovery does not grasp. There is a high insist for mechanical fault localization techniques which can direct programmers to the locations of faults with least human involvement.

This insists has led to the suggestion and development of a variety of techniques over current years. While these techniques split similar goals, they can be moderately diverse from one another, and often stalk from ideas which themselves initiate from some diverse disciplines.

2. FAULT LOCALIZATION

Web applications are written in a mixture of some programming languages, like JavaScript, PHP, and Structure Query Language (SQL). In web application domain, pages are not displayed properly due to the deformed HTML or any cross language errors. Such HTML and cross based language failures may be difficult to find because theses codes are generated dynamically. In this paper, cross based language fault locations are identified by using traverse analysis based technique and heuristic search strategies. Traverse analysis based techniques construct cross tabulation for each executable statement and determine the suspiciousness of the corresponding statement. Heuristic search has been generating dynamic test suites.

In previous work, determined the failures based on the identifying inputs which cause an application hurtle. It did not address the problem. Statistical analysis between passing and failing tests for discover the location of fault. Further many statistical fault localization algorithms are used to find the fault localization process. Using Tarantula [9], [10], Ochiai [1], Jaccard algorithms for finding the percentage of passing and failing tests in the execution of statements. Suspiciousness rating is calculated for each executed statement for forecast the location of the fault. The enhancements are presented in the statistical fault localization techniques for improving efficiency of fault localization. Source mapping method and extended domain are enhanced in our previous work [13]. Extended domain technique applies to conditional statements helps fault localization. Suppose any statements are missing in HTML code or any default case not mention in switch case that time suspiciousness rating cannot be calculate. Condition modelling technique is used in this situation.

2.1 Source mapping

Each statements of web application and output of every part are recorded and mapped. HTML Validator analyses these report and indicate which ingredient of the HTML output are incorrect.

Existing fault-localization approaches assume the existence of a test suite. Though, developers are often tackled with state of affairs where a failure takes place,

but where no test suite is accessible that can be used for fault localization. To address such situations, we present an approach for generating test suites that can be used to localize faults effectively. This approach is a variation on combined concrete and symbolic execution [6], [7] that is parameterized by a similarity criterion. Path constraint similarity and input similarity are increased statement coverage and path coverage. The quantity of similarity among the path constraints related with two executions. Based on the number of inputs the similarity among two execution is figured and it identical for both execution. Automated tool Apollo used for implementing these techniques. It increases the effectiveness of fault localization by direct test suites generation. Each and every dynamic web application contain cross based language, in our previous work does not consider the cross based fault.

3. TRAVERSE ANALYSIS TECHNIQUE

3.1 Objective

The fault-localization algorithms explored in this paper attempt to predict the location of a fault based on a statistical analysis of the correlation between passing and failing tests and the program constructs executed by these tests. In particular, we investigate variations on three popular statistical fault-localization algorithms, known as Tarantula, Ochiai, and Cross tab. These algorithms predict the location of a fault by computing, for each statement, the percentages of passing and failing tests that execute that statement of the client side java script.

In this paper using traverse analysis technique for fault localization in client side JavaScript and compare this performance to Tarantula[9],[10], Ochiai[1] fault localization algorithms. It increases the branch coverage stability and test coverage percentage.

3.2 Traverse Analysis Techniques

Number of faults localization techniques on dynamic analysis of a program is in the form of trace collection. It traced executed statement during program execution. These traced statements have been processed based on the fault localization techniques. In traverse analysis technique has been used the program traces and apply the rank to every executed statement. Traverse tabulation is made for apiece statement and determine the suspiciousness of the corresponding statement. Statements are arranged descending order that is higher suspiciousness placed top of the order and also a higher probability of containing bugs. Traverse analysis consider multiple statements and multi-line statement bugs. A bug should extent multiple statements, when reached fault in the first statement then stop the examination. This process is starting stage of bug fixing process. Multiple faults are different from multi-line fault in the similar program. Single fault communicate to various position in the program. Consider the case where the clauses of an 'if-else' construct are accidentally swapped. In this situation same fault contain 'if' part and also 'else' part. Increase the efficiency of fault localization in the cross based language, have generate dynamic test suites. In previous work combined concrete and symbolic execution techniques have used for dynamic test generation. In our work heuristic search strategies has been used. It is efficient for large scale programs.

It construct table for each executed statement that information are useful for fault fixing process. Table contains three columns and three rows. Columns mention number of cover and uncover statements in the web application during execution and rows represent number of successful and failure of execution. Final column denotes number of test case covering ω and number of test case uncovering ω. Final row mention total number of failed and successful test cases. The cross tabulation is mention below.

Table1: Crosstab for each statement

	covered	uncovered	Σ
Successful execution	$N_{CS}(\omega)$	$N_{US}(\omega)$	N_S
Failure execution	$N_{CF}(\omega)$	$N_{UF}(\omega)$	N_F
Σ	$N_C(\omega)$	$N_U(\omega)$	**N**

The following notations are used for calculating suspiciousness of the executed statement in the web application.

Table 2: Notations used in this paper

N	**Total number of test cases**
N_S	**Total number of failed test cases**
N_F	**Total number of successful test cases**
$N_C(\omega)$	**Number of test cases covering ω**
$N_{CF}(\omega)$	**Number of failed test cases covering ω**
$N_{CS}(\omega)$	**Number of successful test cases covering ω**
$N_U(\omega)$	**Number of test cases not covering ω**
$N_{UF}(\omega)$	**Number of failed test cases not covering ω**
$N_{US}(\omega)$	**Number of successful test cases covering ω**

3.3 System Architecture

Figure 4.1 shows the proposed system architecture in which the function of fault localization process. PHP web application taken as input for fault localization process that produce what are the possible locations for occurring fault. Fault localization process using some algorithms such as Ochiai, Tarantula, and Crosstab that determine the suspiciousness ratings for executed statement. Executed statements are arranged in the order which one

having high suspiciousness rating placed in the top of order. Generate direct test case based on combined concrete and symbolic execution and similarity criteria. Create cross tabulation based on successful and failure execution. Finally compare the performance of different algorithms.

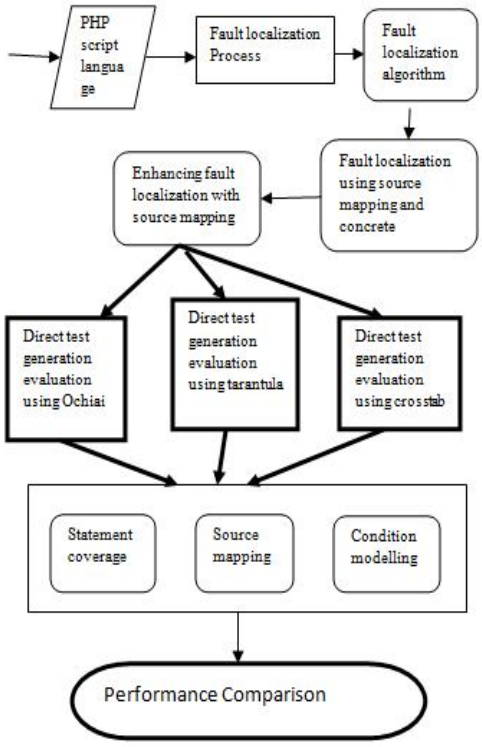


Figure 1 System Architecture

4. DYNAMIC TEST GENERATION

Several techniques are available for dynamic test generation process[13]. Test suites have been generated mainly for cross based language. In this paper, heuristic search strategies are guided by control flow graph of the program under test. CREST open source tool has implemented for dynamic test generation process. Test generation process is increase the efficiency of fault localization process in the cross based language. Concrete and symbolic execution [2], [6], [7] is used for test generation process. It is efficient for smaller programs, these approaches fail to large programs. In this paper focus on search strategy which lead by the static structure of the program beneath test.

4.1 Depth-First Search

The depth-first search first efforts to vigor the first branch by solving path constrain. All feasible program paths have been discovered when search ends. A branch along a path is feasible if we can solve for inputs for which the branch is and is not token. Depth-first search will discover 2^d execution path. All paths have at least d feasible branches.

4.2Control-Flow Directed Search

To lead the dynamic search of the program's path space based on the static structure of the program under test. In this search guide towards paths that reach previously uncovered branches. Control-Flow Directed Search reaches high coverage of the test program. In the stating stage construct control flow graph for every function and provide weight for each edges. Uncovered branches are compute with breadth-first search.

4.3Uniform Random Search

A program is executed on random input, execution of the program through random path. Random search have let alone from many inputs executed through the same path. Uniform random search will produce some particular execution

5. IMPLEMENTATION

We have implemented traverse analysis technique for cross based language fault localization process. Dynamic web application has base language and cross based language. In this paper, have implemented base language fault localization techniques also, and then compare the performance of the fault localization. To increase the efficiency of fault localization process, use CREST tool for dynamic test generation. Create a web application as an input for determine the fault localization. Then implement fault localization tool Apollo and fault localization algorithms such as Ochiai and Tarantula.

This implementation is achieved by the following steps.

- Developing a web application
- Implementation of fault localization in Apollo
- Fault localization based on Ochiai
- Fault localization using Tarantula
- Implementing Traverse technique

5.1 Developing a Web Application

We are developing a web application based on the PHP language which will be given as input. A typical PHP web application is a client/server program in which data and control flow interactively between a server, which runs PHP scripts, and a client, which is a web browser. The PHP scripts generate HTML code, which gets pushed to the client. Such code often includes forms that invoke other PHP scripts and pass them a combination of user input and constant values taken from the generated HTML.

5.2 Implementation of Fault Localization in Apollo

The fault-localization algorithms explored in this module attempt to predict the location of a fault based on a statistical analysis of the correlation between passing and failing tests and the program constructs executed by these tests. In particular, we investigate variations on three popular statistical fault-localization algorithms, known as Tarantula, Ochiai, and Crosstab. A suspiciousness rating is computed for each executed statement. Programmers are encouraged to examine the executed statements in order of decreasing suspiciousness. The effectiveness of a fault localization technique can be measured by determining how many statements need to be inspected, on average, until the fault is found.

5.3 Fault Localization Based on Ochiai

We investigate the influence of the similarity coefficient on the quality of the diagnosis by applying a number of similarity coefficients known from the literature on the same data, and comparing the resulting diagnoses [1]. We investigate the influence of the similarity measure on the quality of the diagnosis. To this end we apply the fault localization technique on a benchmark set of software faults known as the Siemens Suite. We obtain multiple diagnoses for every fault in the suite, each of them for a different similarity measure. These similarity measures are taken from existing diagnosis tools in the areas of recovery and automated debugging, and from the molecular biology domain (Ochiai coefficient). A diagnosis for a particular measure of similarity consists of a list of possible locations for the fault ranked in order of similarity. Our evaluation is based on the position of the (known) location of the fault in this ranking.

5.4 Fault Localization using Tarantula

A fault localization technique is associated with suspiciousness rating to each statement [9]. The statement have high suspiciousness rate that contain maximum fault. The involvement of fault is high in those statements. It calculated based on the total number of successful execution and total number of failure execution of the program for given test case. The statements are arranged in to descending order. This process is fixing the location of the fault in the base language of web application. Cross based language fault fixing is determine by the below method.

5.5 Implementing Traverse Technique

Number of covered and uncovered statements is denoted from the total number of statements during execution. Number of successful and failure executions are also. Using this information determine the suspiciousness by the following formula.

$$\varphi(\omega) = \frac{N_{CF}(\omega)/N_F}{N_{CS}(\omega)/N_S} \qquad (1)$$

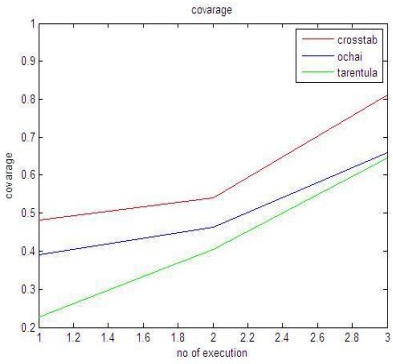
If $\Phi(\omega) = 1$, the execution result is totally autonomous of the coverage of ω. If $\varphi(\omega) > 1$, the coverage of ω is more associated with the failed execution. In case of $\varphi(\omega) > 1$, statement have high suspiciousness. Next case is $\varphi(\omega) = 1$ then least suspiciousness is $\varphi(\omega) < 1$.

6. TOOLS

Number of existing tools is available for dynamic test generation process. Concrete and symbolic execution techniques [2], [6], [7] are useful and efficient for smaller programs. We have been use CREST, grep2.2 and vim 5.7. These tools are efficient for large scale programs. These three tools are open source. CREST tool to extract control-flow graphs and to solve path constraints. It gives absolute branch coverage tool. Grep tool council the control flow search and random branch search. Vim tool also covered most branches.

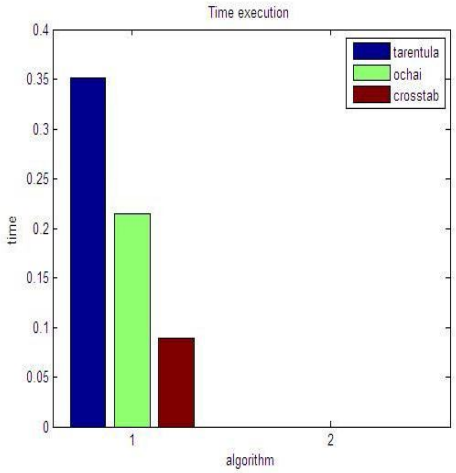
7. RESULT AND DISCUSSION

In this section, discuses about the evaluation result of our concepts. In general the number of statement examined increase the statement coverage of the application also increased. Compare three algorithms, crosstab method having high coverage rate that is total number of statement examined high so that finding location of fault also increased. Ochiai having next high coverage represent in graph. Tarantula having minimum error finding rate that is examine minimum statements examine. X axis represents number of execution of each algorithm and Y axis mentions statement coverage of each algorithm. This increment represents the efficiency of fault localization of the application.



Using an automated tool for testing increasing the efficiency of fault localization process. Speed of the process is increased so reduced the time period. X axis mention three algorithms and Y axis represent time period of the process. Compared to the manual fault localization process, it reduced the time period of localization process. It also reduced finding wrong location of fault. Traverse method shows better performance when compared to that of the existing system, the analysis is carried out in PHP web application

and the results are such as error rate, statement coverage, time and speed. Error rate measure how the algorithm find the wrong location. Compare three algorithms our method having minimum error rate. If there is no test suites available, dynamic test generation method used. Using this method create test suites dynamically, this process reduce testing time and increase speed of the process.



7. RELATED WORK

Literature survey has been done in the area of Software Engineering and its testing process. The research done by various authors are studied and some of them are discussed in the following section.

7.1 Static, Dynamic, and Execution Slice-Based Techniques

Program slicing is a generally used method for debugging. The debugging search domain is reducing by using slice method [5]. Suppose a statement variable having erroneous value then a test case will fail. So the bug must be found in the static slice associated with that variable-statement pair.

It might produce a dice with convinced statements which should not be included that is a drawback of this method. An alternative is to use execution slicing and dicing to locate program bugs, where an execution slice with respect to a known test case contains the set of code executed by this test [8]. There are two principles

• Execute a segment of code with less possible of having some fault that is more successful.

• A piece of code is executed that gives failed test and it contains fault.

The problem of using a static slice is that it discovers statements that could possibly contain a shock on the variables of attention for any inputs in its place of statements that certainly affect those variables for a precise input [11]. Stated in a different way, a static slice does not build any use of the input values that depiction the fault. The main drawback of this approach is take more time and file space for collecting and we recognize the coverage of the test then only easily construct execution slice for a given test.

7.2 Program Spectrum-based Techniques

A program spectrum records the execution information of a program in certain aspects such as how statements and conditional branches are executed with respect to each test [4]. When the execution fails, such information can be used to identify apprehensive code that is accountable for the failure.

The total contribution of the failed tests is larger than that of the successful. The execution of a test is represented as a sequence of basic blocks that are sorted by their execution counts. If an insect is in the distinction set between the failed execution and its most similar successful execution, it is located. For an insect that is not contained in the distinction set, the technique continues by first constructing a program dependence graph, and then including and checking adjacent un-checked nodes in the graph step by step until the bug is located. The set union, and set intersection techniques are also reported. By varying the quality and quantity of the observations on which the fault localization is based, much wider context has been established.

7.3 Statistics-based Techniques

Several statistical fault localization methods are available in advanced fault localization techniques [12]. It does not confine itself to faults located only in predicates. More precisely, across tab is make for each statement with two column-wise definite variables of enclosed, and not enclosed, and two row-wise definite variables of successful execution, and failed execution. The exact suspiciousness of each statement depends on the degree of association between its coverage (number of tests that cover it) and the execution results. More number of approaches is available for fault localization process. Statistics based method is important fault localization method.

7.4 Program State-based Techniques

A program state contain of variables and their values at a meticulous point throughout the execution. The common approach for using program states in fault localization is to modify the values of a number of variables to determine which one is the cause of erroneous program execution. A program state-based debugging approach, delta debugging, to decrease the reasons of failures to a miniature set of variables by complementary program states between executions of a successful test and a failed test by way of their memory graphs. Based on delta debugging, the reason transition technique to identify the locations and times where the reason of failure alter from one variable to another.

A latent problem is that the cost is relatively high; there may exist thousands of states in a program execution, and delta debugging at every corresponding point need additional test runs to slight the reason. Another problem is that the recognized locations may not be where the bugs reside.

7.5 Machine Learning-based Techniques

Machine learning techniques are adaptive, and vigorous; and have the capability to make models based on data, with limited human interaction. The problem at offer can be uttered as trying to be taught or assume the location of a fault based on input data such as statement coverage, etc. The statement coverage of every test case, and the equivalent execution result, are used to train a BP neural network.

Then, the reporting of a set of virtual test cases that all covers only one statement in the program are input to the qualified BP network, and the outputs can be regarded as the probability of the statements being faulty. on the other hand, as BP neural networks are recognized to suffer from problems such as paralysis, and local minima.

The statement coverage of both the failed and successful test cases in each partition is then used to form a ranking based on every panel using a heuristic similar. These entity rankings are then combined to figure a concluding statement ranking which can then be examined to locate the faults.

8. CONCLUSION AND FUTURE WORK

In this paper, we introduce a fault localization approach for cross based language in web application. This process is achieved on traverse analysis technique. Traverse analysis process determined the suspiciousness of the executed statement in the cross based language on web application.

In future, we plan to increase the efficiency of fault localization process in cross based language.

ACKNOWLEDGEMENT

First and foremost, I thank the Lord Almighty who paved the path for my walk which lifted me to pluck the fruits of success and who is the torch for all my endeavours and engagements.

I wish to express my deep sense of gratitude and hearted thanks to IJETTCS, for selected my paper work to be publish International Journal.

I record my deep sense of indebtedness and whole heated gratitude to my friends, S.ASHOK KUMAR, M.ISAIKKANI, M.SIVALINGAM, for their active involvement, encouragement, caretaking and valuable suggestions in all the way and also shaping project work and include report.

Home is the backbone of every success and our humble salutations to our beloved parents who inspired, motivated and supported us throughout the course of the project.

I also extend my sincere thank to all the Faculty Members of CSE Department, Friends who have render their valuable help in completing this project successful.

REFRENCES

[1] R. Abreu, P. Zoeteweij, and A.J.C. van Gemund, “An Evaluation of Similarity Coefficients for Software Fault Localization,” Proc. 12th Pacific Rim Int’l Symp. Dependable Computing, pp. 39-46, 2006.

[2] C. Cadar, V. Ganesh, P.M. Pawlowski, D.L. Dill, and D.R. Engler, “EXE: Automatically Generating Inputs of Death,” Proc. Conf.Computer and Comm. Security, 2006.

[3] B. Baudry, F. Fleurey, and Y. Le Traon, “Improving Test Suites for Efficient Fault Localization,” Proc. 28th Int’l Conf. Software Eng.,L.J. Osterweil, H.D. Rombach, and M.L. Soffa, eds., pp. 82-91,2006.

[4] R. Abreu, P. Zoeteweij, and A.J. van Gemund, “On the Accuracy of Spectrum-Based Fault Localization,” Proc. Testing: Academic and Industry Conf. Practice and Research Techniques, pp. 89-98, Sept. 2007.

[5] H. Agrawal, J.R. Horgan, S. London, and W.E. Wong, “Fault Localization Using Execution Slices and Dataflow Tests,” Proc. Int’l Symp. Software Reliability Eng., pp. 143-151, 1995.

[6] P. Godefroid, N. Klarlund, and K. Sen, “DART: Directed Automated Random Testing,” Proc. ACM SIGPLAN Conf. Programming Language Design and Implementation, 2005.

[7] P. Godefroid, M.Y. Levin, and D. Molnar, “Automated Whitebox Fuzz Testing,” Proc. Symp. Network and Distributed System Security,2008.

[8] S. Horwitz, T. Reps, and D. Binkley, “Interprocedural Slicing Using Dependence Graphs,” ACM Trans. Programming Languages and Systems, vol. 12, no. 1 pp. 26-60, 1990.

[9] J.A. Jones and M.J. Harrold, “Empirical Evaluation of the Tarantula Automatic Fault-Localization Technique,”Proc. IEEE/ACM Int’l Conf. Automated Software Eng., pp. 273-282, 2005.

[10] J.A. Jones, M.J. Harrold, and J. Stasko, “Visualization of Test Information to Assist Fault Localization,” Proc. Int’l Conf. Software Eng., pp. 467-477, 2002.

[11] J. Lyle and M. Weiser, “Automatic Bug Location by Program Slicing,” Proc. Second Int’l Conf. Computers and Applications, pp. 877-883, 1987.

[12] Y. Minamide, “Static Approximation of Dynamically Generated Web Pages,” Proc. Int’l Conf. World Wide Web, 2005.

[13] P. Arumuga Nainar and B. Liblit, “Adaptive Bug Isolation,” Proc.32nd ACM/IEEE Int’l Conf. Software Eng., pp. 255-264, 2010.

[14] G.K. Baah, A. Podgurski, and M.J. Harrold, “Causal Inference for Statistical Fault Localization,” Proc.

19th Int'l Symp. Software Testing and Analysis, pp. 73-84, 2010.
[15] M.Y. Chen, E. Kiciman, E. Fratkin, A. Fox, and E. Brewer, "Pinpoint: Problem Determination in Large, Dynamic Internet Services," Proc. Int'l Conf. Dependable Systems and Networks, pp. 595-604, 2002.
[16] T.M. Chilimbi, B. Liblit, K.K. Mehra, A.V. Nori, and K. Vaswani, "Holmes: Effective Statistical Debugging via Efficient Path Profiling," Proc. 31st Int'l Conf. Software Eng., pp. 34-44, May 2009.
[17] H. Cleve and A. Zeller, "Locating Causes of Program Failures," Proc. Int'l Conf. Software Eng., pp. 342-351, May 2005.
[18] V. Dallmeier, C. Lindig, and A. Zeller, "Lightweight Defect Localization for Java," Proc. European Conf. Object-OrientedProgramming,pp.528-550,2005.
[19] P. Godefroid, N. Klarlund, and K. Sen, "DART: Directed Automated Random Testing," Proc. ACM SIGPLAN Conf. Programming Language Design and Implementation, 2005.
[20] P. Godefroid, M.Y. Levin, and D. Molnar, "Automated Whitebox Fuzz Testing," Proc. Symp. Network and Distributed System Security,2008.

AUTHOR 1

G.Maheswari
Full Time ME Student,
Regional Centre, Coimbatore.
Anna University:: Chennai.

Address

3/24, Gandhi street,
Kaluneerkulam(P.O),
Alangulam (T.K),
Tirunelveli (D.T)
TamilNadu (S.T)
India
Pin Code-627861

AUTHOR 2

Dr.V.Venkatesakumar
Assistant HOD of CSE Department
Regional Centre, Coimbatore.
Anna University:: Chennai.