IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Vol.-7, Issue-3, March 2019 E-ISSN: 2347-2693

An Effective Search Based Algorithm for Structural Test Data Generation

Sachin D. Shelke[1*], S.T. Patil[2]

[1,2]Department of Computer Engineering, Vishwakarma Institute of Technology, Pune, India

[*]*Corresponding Author: sacheenshelke@gmail.com, Tel.: +91-9975614005*

DOI: https://doi.org/10.26438/ijcse/v7i3.517522 | Available online at: www.ijcseonline.org

Accepted: 14/Mar/2019, Published: 31/Mar/2019

***Abstract*—** Software testing is process for improving the quality of software by removing all sorts of errors before deployment of software system. The quality of the testing also depends on the test data used for the testing. If the test data cover all the statements and branches of a source program, then it increases the chances of revealing most of the errors from the given program. Normally test data is selected by tester based on his past experience of similar projects. This is time consuming and person oriented approach. Automation of this process can make the testing efficient, cost-effective and reliable. So we present here the Effective Search Based Algorithm (ESBA) which automatically generates test data to reveal the errors at structural test. Here we used branch distance as the optimization function to generate the test data. We applied this method on three benchmark programs to generate the test data. The experimental results indicate that our method outperforms genetic algorithm, many objective sorting algorithm based upon following criteria: average statement coverage 0.91, average branch coverage 0.84 and the average number of evaluations 23824.

***Keywords*—**Automated Software Testing, Automated Test Data Generation, Structural Testing,, Search based Algorithm, Branch Coverage

I. INTRODUCTION

Software testing has been the primary way for assuring high quality of software systems. Software testing which according to [1] is the "process of ensuring that a certain piece of software item fulfills its requirements" is one of the vital factors that can ensure the reliability of the software. According to [2] assurance of software reliability partially depends on testing. However, testing itself also needs to be reliable. In software testing, testers need to design a set of test cases, including test inputs and expected outputs, to cover most of the code and find most bugs before releasing the software. It is challenging to generate such effective test cases manually for complex software systems nowadays. Automatic software testing reduces the laborious human efforts in testing. It basically follows three steps approach. Test Data generation, Test execution, and test output inspection. Effectiveness of testing depends upon the test data selected for the testing. Selection of test data that could cover all the statements and branches of a given module is a complex task. So here we proposed the Effective Search Based Algorithm (ESBA) to generate the test data which could be used to test the system effectively.

The use of MHS algorithms for test case generation is referred to as search-based software testing. Mantere and Alander [3] discuss the use of MHS algorithms for software testing in general and McMinn provides a survey of some of the MHS algorithms that have been used for test data generation. The most common MHS algorithms that have been employed for search-based software testing are evolutionary algorithms, simulated annealing, hill climbing, ant colony optimization, and particle swarm optimization. Among these algorithms, hill climbing (HC) is a simpler, local search algorithm. The SBST techniques using more complex global MHS algorithms are often compared with test case generation based on HC and random search to determine whether their complexity is warranted to address a specific test case generation problem. In our proposed ESBA we call the search method in order to minimize the distance at the branch that deviates the program execution from the selected path.

Rest of the paper is organized as follows; section II contains related previous work regarding the automated test data generation techniques. In the section III working of ESBA is given. Results are discussed in the section IV followed conclusion.

II. RELATED WORK

Lots of researchers have tried to atomize the derivation of test cases from specifications. A formal specification like UML based approach for test data generation, symbolic execution, data flow based are commonly used approaches mentioned in [1, 3, 4, 5 and 6]. Apart from formal

specification we also have random test data generation technique and goal-oriented approach for data generation mentioned in [4]. Different algorithms like search based are used to generate test cases from specification approaches. A major research area in this domain is the application of MHS algorithms to test case generation. MHS algorithms are a set of generic algorithms that are used to find optimal or near-optimal solutions to problems that have large complex search spaces. There is a natural match between MHS algorithms and software test case generation. The process of generating test cases can be seen as a search or optimization process [7]. There are possibly hundreds of thousands of test cases that could be generated for a particular SUT and, from this pool, we need to select, systematically and at a reasonable cost, those that comply with certain coverage criteria and are expected to be fault revealing, at least for certain types of faults. Hence, it needs to reformulate the generation of test cases as a search that aims at finding the required or optimal set of test cases from the space of all possible test cases. When software testing problems are reformulated into search problems, the resulting search spaces are usually very complex, especially for realistic or real-world SUTs. For example, in the case of white-box testing, this is due to the nonlinear nature of software resulting from control structures such as selection statements and loops. In such cases, simple search strategies may not be sufficient and global MHS algorithms may, as a result, become a necessity, as they implement global search and are less likely to be trapped into local optima [7, 8, 9, and 10].

There are some good survey papers on test data generation techniques. Shahid Mahmood [11], in his master thesis, he provides a systematic review that is aimed at presenting a fair evaluation of research concerning ATDG techniques of the period 1997-2006. Moreover, it aims at identifying probable graphs in the research about ATDG techniques of defined period so as to suggest scope for further research. Phill Macminn, surveys the application of meta-heuristic search techniques to software test data generation. It provides the survey of structural, functional and nonfunctional test data generation techniques and research focus in these areas. Saswat Anand [12], provides an orchestrated survey of methodologies for automated software test data generation. The review consists of a brief description of the basic ideas underlying the technique, a survey of the current state of the art in research and practical use of the techniques, a discussion of remaining problems for the further research and a perspective of the future development of the approach. Corina Pasareanu [13], surveys new techniques based on symbolic execution and some of their traditional applications, such as test data generation and program analysis as well as some new, interesting applications.

Based on earlier studies it is clear that test data generation is crucial and of interest topic of many researchers, which plays an important role in deciding the effectiveness of testing strategy. In the next section we have discussed our ESBA structural test data generation process followed by evaluation parameters used to compare the different test data generation techniques with ESBA. Experimental results are discussed in the last section of this paper followed by a conclusion.

III. METHODOLOGY

A. Generating Test Data Using ESBA

In this paper, ESBA is proposed to generate the test data. Figure 1. illustrates the working of the proposed method. In the proposed method first program is selected. We used the three benchmark programs Table 2. for which we generate the test data. After program selection, we set the parameters that are used for evaluation like branch coverage, statement coverage etc.

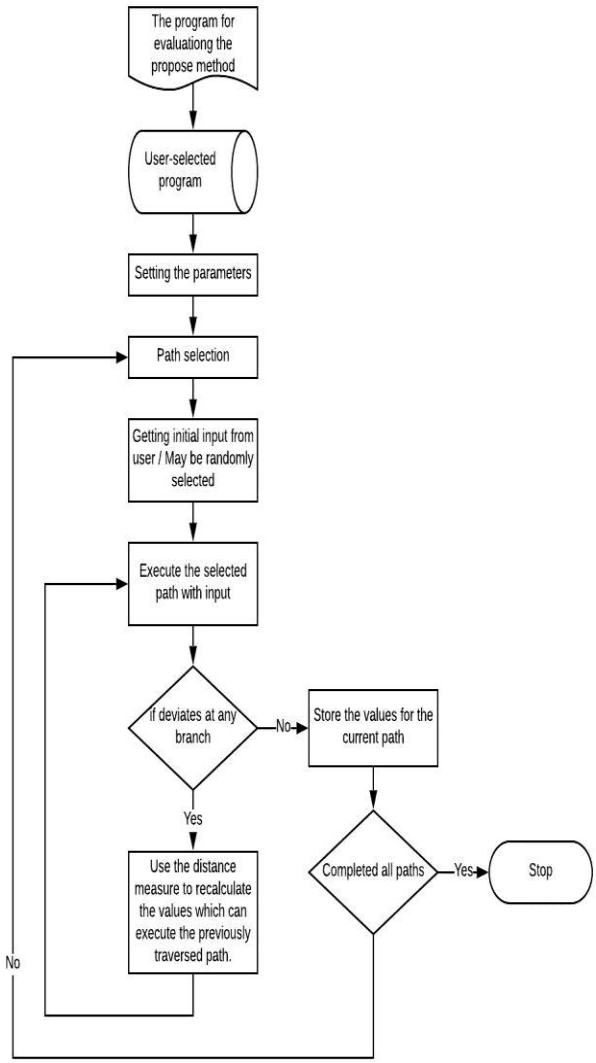

Figure 1. The diagram of proposed method (ESBA)

The program may have multiple paths in this method we select a single path at a time. Test data for the selected path will be generated. Then we select the next path and repeat the same process until we cover all the paths.

In the test data generation process, tester selects a path through the program, and then produces a straight-line version of it, containing only that path. Then we replace branching statements with a "path constraint" of the form ci = 0; ci > 0; or ci ≥ 0; where ci is an estimate of how close the constraint is to being satisfied. For example, a branch predicate of the form a == b gets rearranged as a path constraint abs(a − b) = 0. Using these constraints, a function f is constructed. The value of f provides a real-valued estimate of how close all of the constraints are to being satisfied, being negative when one or more of the constraints remain unsatisfied, and positive when all of the constraints are satisfied. Initially, input values are randomly selected or these values may be specified by a tester, then these values are readjusted using search techniques to satisfy the constraints. When violation at any branch for selected path occurs then search function is called. It takes the branch distance as a measure and adjusts the input values in a way that the distance becomes zero or negative in order to execute the branch. Values are updated in such a way that it evaluates the current branch to true and traverse the already traversed path again.

B. Branch Predicate

Table 1. shows the branch predicates to be replaced by branch function of form F rel 0 in order to generate the test data.

Table 1. Branch predicate and Equivalence Predicate of Form F rel 0

Branch Predicate	Branch Function	rel
E1 > E2	E2 - E1	<
E1 ≥ E2	E2 - E1	≤
E1 < E2	E1 – E2	<
E1 ≤ E2	E1 – E2	≤
E1 = E2	abs (E1 – E2)	=
E1 ≠ E2	abs (E1 – E2)	≤

After the path selection, straight line version of the path is generated. It is done using the branch function of form F rel 0 given in Table 1. All the branch predicates are replaced with the function of type F rel 0. The value of function provides the estimate of how close the constraint to being satisfied. According to the function the input values are changed using the search technique.

IV. RESULTS AND DISCUSSION

A. Benchmark Programs

We used commonly used three benchmark programs here, triangle Type Program, calDay Program and calnDays to generate the test data. We applied Genetic Algorithm, MOSA [18] and our algorithm on each of these programs for generation of test data.

First program triangle Type states the type of Triangle. It takes three real numbers as an input for different triangle edges and states whether given Triangle is an equilateral triangle, an isosceles triangle or a scalene triangle.

Table 2. Benchmark Programs for Automated Test Data Generation

Program	#Args	#Arg Type	LOC	Description
triangleType	3	Integer	45	Type classification for for a triangle
calDay	3	Integer	45	Calculate the day of week
calnDays	6	Integer	255	Compute the days between two dates

The second program calDay states the day of the week on the input of a specific date. The date is entered as the three integer values stating a day, a month and a year. Third program takes six integer parameters as input, the first three parameters comprising of a start date and next three parameters comprising of an end date, to return back the number of days between the specified dates as an output. Line of code for each program is also given in the above Table 2.

B. Evaluation Parameters

In order to decide the effectiveness of the testing technique, we need to evaluate the applied technique with the help of certain parameters such as code coverage, number of iterations etc.

1. Statement Coverage

Every statement of the source code /program is executed at least once then we can say that we achieved the 100% statement coverage. Statement coverage helps us to identify the typographical errors, logical errors etc. Though statement coverage is not an effective measure for deciding the effectiveness of the testing method, it can be definitely used for analysis purpose. Statement coverage is also useful in deciding the unreachable code of the module/program. In our method Effective Search Based (ESB) algorithm, we nearly achieved 100% statement coverage for three benchmark programs.

2. Branch Coverage

Branch coverage criteria require enough test cases such that each point of entry to a program or subroutine is invoked at

least once. That is, every branch (decision) taken each way, true and false. It helps to validate all the branches in the code making sure that no branch leads to abnormal behavior of the application. Using ESB it is shown that the more branch coverage is achieved for all three benchmark programs than the GA and MOSA.

3. Number of Evaluations

The Number of Evaluations is nothing but the number of runs used by an algorithm/method in order to find the test data. This measure is helpful in order to compare the methods to decide which method takes large number of evaluations. As number of evaluations are directly proportional to the time required to generate the data.

OTHER PARAMETERS:

Following are the other parameters; these can also be used for comparison purpose.

1. Success Rate (SR)

It refers to the probability of coverage of all the available branches in the program via the generated test data. In this criterion, the output with higher values stands for better performance.

2. Average Time (AT)

Average time, refers to the time at which all branches are covered. Time is measured in milliseconds (ms). Here, the output with lower values indicates a better performance.

C. Experimental Results and Discussion

Table 3. shows the comparative experimental results for three benchmark programs. We used the statement coverage, branch coverage and number of evaluations, these three evaluation parameters for comparing GA, MOSA and proposed ESB algorithm. We also plotted the same results using bar chart and pie charts for comparison purpose.

Table 3. Comparison chart for MOSA, GA and Proposed ESB

		MOSA			GA			ESB		
Test Object	#Branches	Coverage			Coverage			Coverage		
		Statement	Branch	#Evaluations	Statement	Branch	#Evaluations	Statement	Branch	#Evaluations
triangleType	22	0.900	0.808	29,451	0.976	0.892	35,444	1.000	0.917	33,021
calDays	38	0.748	0.634	9,763	0.750	0.636	11,777	0.755	0.644	9,972
calnDays	145	0.750	0.608	19,356	0.972	0.968	28,428	0.981	0.978	28,481

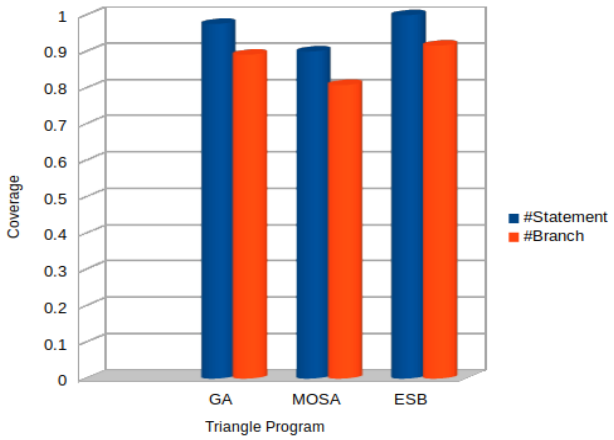

Figure 10. Statement Coverage and Branch Coverage for Triangle Program

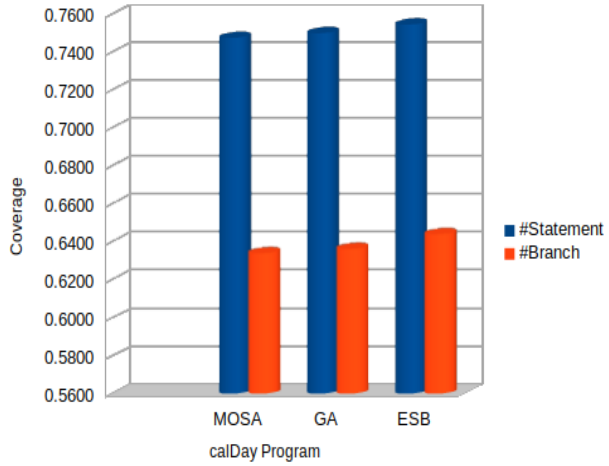

Figure 11. Statement Coverage and Branch Coverage for calDays Program

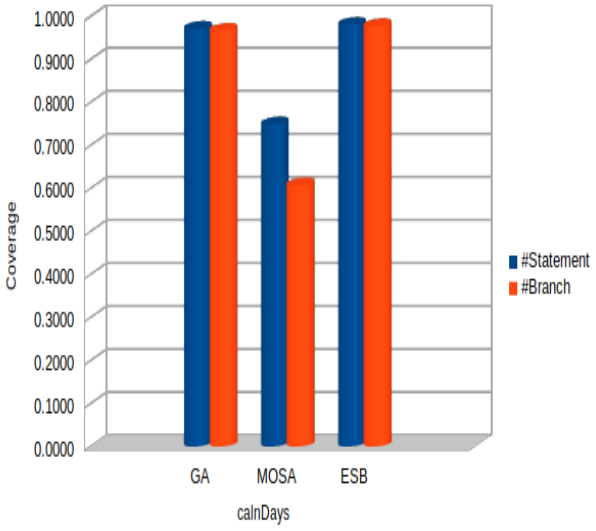

Figure 12. Statement Coverage and Branch Coverage for calnDays Program

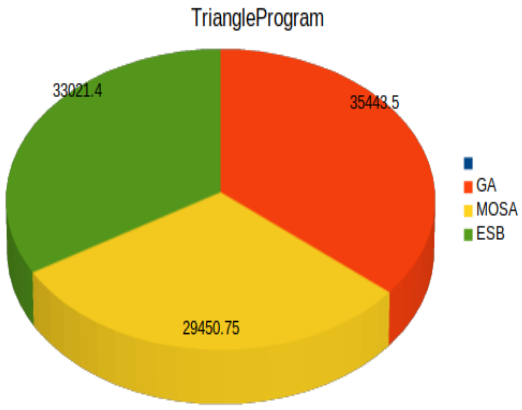

Figure 13. Number of Evaluations for Triangle Program

Figure 10 shows the statement and branch coverage for triangle program. Figure 11 shows the statement and branch coverage for a calDay program, while Figure 12 shows the statement and branch coverage for a calnDays program. From Figure 10, 11 and 12 we can see that the proposed effective search-based method gives the better statement and branch coverage for all programs. Next parameter used for comparison is the number of evaluations it takes for finding the test data. The number of evaluations for used benchmark programs is shown in Figure 13, 14 and 15.

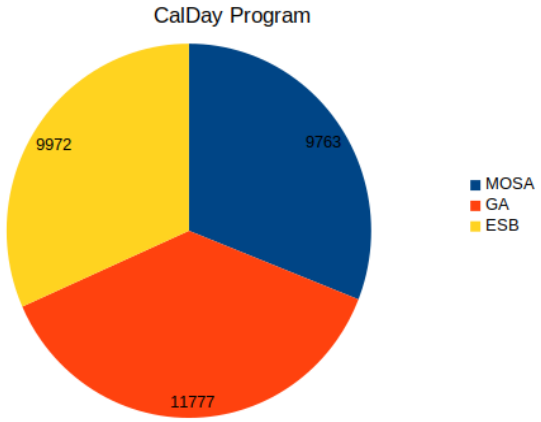

Figure 14. Number of Evaluations for calDay Program.

From the pie charts we can observe that the number of evaluations taken in order to find the test data is lesser than the other methods, GA and MOSA for triangle program and calDay program. For calnDays program, though the number of evaluations taken to find test data is higher than MOSA, it is clear from Figure 15. that it is due the fact that in many runs MOSA was unable to find the test data and terminated the runs earlier without covering all the statements from the source program.

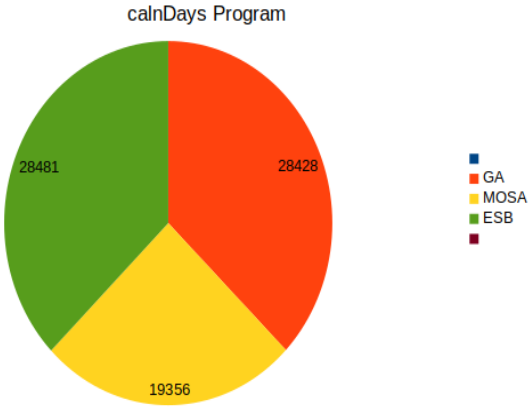

Figure 15. Number of Evaluations for calnDays Program.

Statement coverage, branch coverage and the Number of Evaluations are the parameters used for comparing the different methods. We have compared these methods for three benchmark programs stated in Table 3. ranging LOC from 45 to 255 and branches ranging from 22 to 145. Statement coverage gives you the idea regarding how many statements (% percentage) from the given program are covered by the generated test data. Though the statement coverage is not considered as the effective measure for test data generation, but is important for comparison purpose when we are evaluating the different methods. Most commonly used evaluation parameter is branch coverage, along with branch coverage we may have the condition coverage but it increases the complexity level and may turn into higher number of evaluations. The Number of evaluations is the third parameter used for comparison. From above data we can clearly state that the proposed ESB algorithm gives the better statement and branch coverage for all three benchmark programs with less number of evaluations.

V. CONCLUSION AND FUTURE SCOPE

Software testing is time-consuming process which may costs up to 50 % of the overall software development cost. Careful selection of test data will help to reduce the efforts required in testing. Selection of test data basically depends upon the testers' experience, knowledge regarding the input domain of similar kind of projects. So at large it is an art and depends upon the testers' skill. Automation of this process helps to the dependence and gives a systematic way to automatically generate the test data which will help to reduce the efforts required for testing. Currently there hardly any single concrete technique that can be used to generate the test data for industrial applications. Many of the techniques available to generate the test data for the structural test are applied on the test examples, and may not be applicable for industrial applications. Here we used search-based method proposed (ESB) in order to generate the test data that is used to traverse

the internal structure. The results are analyzed and evaluated based upon following three criteria; statement coverage, branch coverage, and the number of evaluations. The proposed method is compared with MOSA and GA. The experimental results indicate that our method outperforms genetic algorithm and many objective sorting method based on following criterion: average statement coverage 0.91, average branch coverage 0.84 and number of evaluations 23824.

Further other parameters like success rate (SR) and average time (AT) can also be used for comparing proposed ESB with MOSA and GA. Moreover, we can also compare proposed ESB with random search as it is a simple but effective test technique and used by many researchers for comparison purpose.

REFERENCES

[1] Shaukat Ali, Muhammad Zohaib Iqbal, Andrea Arcuri, and Lionel C. Briand, "Generating Test Data from OCL Constraints with Search Techniques", IEEE Transaction on Software Engineering, Vol. **39**, NO. **10**, **October 2013**.

[2] Mark Harman and Phil McMinn, "A Theoretical and Empirical Study of Search-Based Testing: Local, Global, and Hybrid Search", IEEE Transaction on Software Engineering, Vol. **36,** NO. **2, March/April 2010**.

[3] T. Mantere and J.T. Alander, "Evolutionary Software Engineering, a Review," Applied Soft Computing, vol. **5,** pp. **315-331**, **2005**.

[4] Roy P. Pargas, Mary Jean Harrold, Robert R. Peck, "Test Data Generation using Genetics Algorithms", Journal of Software Testing, Verification and Reliability, **1999**.

[5] Gilles Bernot, Marie Claude Gaudel, Bruno Marre, "Software Testing based on Formal Specifications:a theory and a tool", Software Engineering Journal (SEJ), Vol.**6,** No-**6**, p.3**87-405**, **1991**.

[6] Sandra Rapps and Elaine J. Weyuker, "Selecting Software Test Data Using Data Flow Information", IEEE Transactions On Software Engineering, Vol. SE-**1l**, No. **4**, **April 1985**.

[7] Phil McMinn, "Search-based Software Test Data Generation:A Survey", Software Testing, Verification and Reliability **14(2)**, pp. **105-156,** June **2004**.

[8] Mark Harman and Phil McMinn, "A Theoretical and Empirical Study of Search-Based Testing: Local, Global, and Hybrid Search", IEEE Transaction on Software Engineering, Vol. **36**, NO. **2**, **March/April 2010**.

[9] Hwa-You Hsu and Alessandro Orso, "MINTS: A General Framework and Tool for Supporting Test-suite Minimization", IEEE ICSE'09, **May 16 - 24, 2009**, Vancouver, Canada.

[10] Christoph C. Michael, Gary McGraw, and Michael A. Schatz, "Generating Software Test Data by Evolution", IEEE Transaction on Software Engineering, Vol. **27**, No. **12**, **December 2001**.

[11] Shahid Mahmood, "A Systematic Review of Automated Test Data Generation Techniques", Mater Thesis, Software Engineering MSE-2007:**26**, **October 2007**.

[12] Saswat Anand, Edmund K. Burke, Tsong Yueh Chen, John Clark, Myra B. Cohen, Wolfgang Grieskamp, Mark Harman, Mary Jean Harrold, Phil McMinn, "An Orchestrated Survey Of Methodologies For Automated Software Test Case Generation", Elsevier, **April 2013**.

[13] Corina S.Pasareanu and Willem Visser, " A survey of new trends in symbolic execution for software testing and analysis", Springer-Verlag- 2009.

[14] Lionel Briand, Yvan Labiche, "A UML-Based Approach to System Testing", Software Quality Engineering Laboratory, Systems and Computer Engineering Department, Carleton University, **2002.**

[15] Shaukat Ali, Lionel C. Briand, Hadi Hemmati, Rajwinder K. Panesar-Walawege, "A Systematic Review of the Application and Empirical Investigation of Search-Based Test Case Generation", IEEE Transaction on Software Engineering, Vol. **36**, NO. **6**, **November/December 2010**.

[16] Annibale Panichella, Fitsum Meshesha, Paolo Tonella, "Automated Test Case Generation as a many-Objective Optimization Problem with Dynamic Selection of the Targets", IEEE **2018**.

[17] Bogdan Korel, "Automated Software Test Data Generation", IEEE Transactions on Software Engineering, **August 1990**.

[18] Simone Scalabrino, Giovanni Grano, Darrio Di Nucci, Rocco Oliveto, and Andrea De Lucia, "Search-based Testing of Procedural Programs: Iterative Single-Target Approach?", Conference Paper **October-2016**.

[19] Zoreh Karimi Aghdam and Bahman Arasteh, "An Efficient Method to Genearate Test Data for Software Structural Testing Using Artificial Bee Colony Optimization Algorithm", International Journal of Software Engineering and Knowledge Engineering Vol-**27**, No-**6**, **2017**.

[20] Simone Scalabrino, Giovanni Grano, Darrio Di Nucci, Michele Guerra, Andria De Lucia, Harald C Gall and Rocco Oliveto, "OCELOT: A Search Based Test Data Generation Tool for C", An International Conference on Automated Software Engineering **ASE'18**.

Authors Profile

Mr. Sachin D. Shelke pursed Bachelor of Computer Engineering from Shivaji University Kolhapur, Maharashtra in 2004 and Master of Information Technology from Bharti Vidyapeeth, Pune in year 2011. He is currently pursuing Ph.D. and working as Assistant Professor in Department of Information Technology, PICT, Pune since 2007. He is a member of Computer Society of India (CSI) since 2007. His main research work focuses on Software Testing, E-commerce Security. He has 14 years of teaching experience.

Mr. S. T. Patil pursed Bachelor of Electronic Engineering, Master of Computer Engineering and Ph. D. in Computer Engineering. He is working as a Professor, Department of Computer Engineering, Vishwakarma Institute of Technology, Pune. He has published more than 65 papers in reputed international journals. He has written nine books and has one patent to his credit. His main research work focuses on Digital Signal Processing, Image Processing, Neural Networks, Computer Networks, Mobile Computing, Microprocessor & Micro-controllers, Embedded Systems, Biomedical Engineering and Bio-informatics.

