International Journal of Computer Applications (0975 – 8887)
Volume 155 – No 8, December 2016

Performance Analysis of Hybrid Approach Comprising Genetic Algorithm and Adaptive Approach on Test Case Prioritization

Rajanroop Walia
M.Tech, CSE
DAVIET, Jalandhar

Harpreet K. Bajaj
Head of Deptt. CSE
DAVIET, Jalandhar

ABSTRACT

Regression testing is an important domain of software testing, which attempts to verify all the fixes that had been introduced into the software throughout its development period by means of test suites. In spite of being exorbitant in terms of time and cost, it cannot be evaded. As a result, lot many techniques have been proposed in the past in order to minimize these expenses. One such technique is Test Case Prioritization, which works by scheduling the execution order of test cases with a goal of improving the fault detection rate. This paper introduces a hybrid approach to test case prioritization, by combining Genetic Algorithm and Adaptive approach. Initially, it applies the Adaptive approach for the prioritization of test cases. Further, the left over test cases are prioritized by applying the Genetic Algorithm. Finally, the outcomes obtained from the proposed approach are compared with those of Genetic Algorithm based on two parameters: execution time and average percentage of statement coverage (APSC) values. The evaluation results prove that the proposed approach performs better in terms of both the parameters.

Keywords

Regression testing, test case prioritization, genetic algorithms, adaptive approach

1. INTRODUCTION

One of the major goals of regression testing is to make sure whether the software under development still performs in the same manner as it did before the modifications were introduced [1]. Nevertheless, it is an expensive process as far as cost and time factors are concerned. This drawback of regression testing stimulated the efforts to cut down these expenses and consequently led to the development of three major techniques: Test Case Prioritization, Test Case Selection and Test Suite Minimization. Test Case Prioritization schedules the execution order of test cases such that the rate of fault detection is improved. Test Case Selection attempts to choose a subset of the original test suite. In case of Test Suite Minimization, the original test suite gets reduced to a smaller suite that still maintains the coverage. Amidst these techniques, Test Case prioritization is known to be most effective. This is so because it takes into consideration all the test cases contained in a test suite and detects the best test case execution sequence that meets a certain testing criteria. This does not happen in case of other two techniques since they do not take account of all the test cases present in a test suite and thus increase the chances of the software containing undetected errors [2].

Test Case Prioritization has been performed in the past using several approaches. Prominent among these are genetic algorithm, particle swarm optimization, ant colony optimization, bee colony optimization, history-based approach and adaptive approach. Genetic Algorithms provide excellent solutions to prioritization problems and thus are widely used. But these prove to be quite time-consuming in case of bigger test suites, since they perform test suite prioritization and execution as separate phases. In contrast, an adaptive approach saves time by carrying out both these processes concurrently. Therefore, it is gaining popularity these days. But it does not schedule the execution order of all the test cases. Rather, it schedules the order of some selected test cases that have attained some amount of statement coverage in the past. Thus it does not prioritize all the test cases, which implies that statement coverage has not been done perfectly. As a result, a hybrid approach has been proposed in this paper. It combines the above two approaches in such a manner that both the approaches counteract the limitations of each other. In this way, the proposed approach achieves almost 100% statement coverage in minimum time.

This paper is organized as follows. Section 2 describes related work. Section 3 gives an insight into some of the existing test case prioritization approaches. Section 4 presents the proposed work. Section 5 explains how the experiment is carried out and presents the results. Section 6 concludes the paper and mentions the future scope.

2. RELATED WORK

A detailed insight into the regression testing practices was carried out in order to resolve the issues associated with it. In [1] Y. Li gave a thorough explanation of regression testing, involving its definition and types. In addition, they also compared the *retest all* and *selective* regression testing strategies and arrived at the conclusion that a tradeoff exists between the both. However, [2] explained that with an increase in the size of test suite, *retest all* strategy becomes impracticable due to time and cost constraints. Thus, it unveiled an increasing trend towards the different strategies for removing these constraints namely, test case prioritization, test suite minimization and test case selection. However, test case prioritization gained much importance which is apparent from the large amount of work that has been done in this area. Y.C. Huang in [3] proposed a cost-cognizant prioritization technique that arranged test cases in accordance with their history information using genetic algorithm. The technique performed prioritization of test cases based on their test costs and fault severities, without examining the source code. The efficiency of the same was evaluated using a UNIX utility program and the results proved the effectiveness of the proposed technique. In [4], a technique for locating the test path to be tested first in case of static testing was proposed. Test paths or scenarios were extracted from the source code. For finding out the path to be tested first, the approach used

Information Flow model and Genetic Algorithm. In [5], the necessity of Component-Based Software testing prioritization framework was evolved and proved, which exposed more severe bugs at an early level and improved software product deliverable quality employing Genetic Algorithm with java decoding technique. For this purpose, a set of prioritization keys was proposed. An algorithm for prioritizing test cases based on total coverage using a modified genetic algorithm was designed in [6]. Its performance was compared with five other approaches and the results revealed the proposed algorithm to be better than other approaches. However, the same could not be assured for bigger test suites. L. Ramingwong in [7] proposed the standard ABC algorithm for prioritizing the test suites based on code coverage. The results revealed that ABC shows promising results and hence, is a great candidate for prioritizing test suites. It also suggested that by modifying the standard ABC algorithm or combining it with another SI algorithm should yield an even better result.

In [8], Y. Singh proposed a regression test prioritization technique based on Ant Colony Optimization for reordering the test suites in time constrained environment. On the other hand, K. Solanki in [9] presented an enhanced version of Ant Colony Optimization for test case prioritization. The metric used for performance evaluation in both the cases was Average Percentage of Faults detected (APFD) metric and the results confirmed the usefulness of these techniques. Tyagi in [10] proposed a 3-step approach for performing regression testing using Multi Objective Particle Swarm Optimization. The proposed MOPSO outshined other approaches like No Ordering, Reverse Ordering and Random Ordering by achieving maximum fault coverage and maximum APFD value in minimum execution time. T. Noguchi in [11] proposed a framework to prioritize test cases for black box testing on a new product using the test execution history collected from a similar prior product and the Ant Colony Optimization. The effectiveness and practicality of the proposed framework was shown by a simulation using two actual products. In [12], history-based approach to prioritize the test cases was extended to modified lines. Initially, the modified lines were prioritized, followed by the test cases. The results proved the proposed approach's capability to detect faults faster and with less effort as compared to previous approach. Md. Arafeen in [13] proposed adaptive regression testing (ART) strategies that attempt to identify the regression testing techniques that will be the most cost-effective for each regression testing session according to organization's situations and testing environment. For assessing the approach, an experiment was conducted by focusing on test case prioritization techniques. The results showed that prioritization techniques selected by the approach can be more cost-effective than those used by the control approaches. Dan Hao in [14] proposed an adaptive TCP approach, which worked by determining the execution order of test cases simultaneously during their execution on the modified program. The results indicated the superiority of the proposed approach over the total test case prioritization approach. It also concluded the proposed approach to be comparable to additional statement-coverage based test case prioritization approach. In [15], L. Mei proposed Preemptive Regression Testing (PRT), a novel strategy that rescheduled test cases according to the changes detected in the service under test, during the course of each actual regression test session. For generating new techniques, three particular PRT strategies, integrated with existing test case prioritization techniques were proposed. The experimental results proved the capability of one of the PRT-enriched techniques to test workflow-based web service. In [16] the existing strategies were empirically studied and two additional Adaptive Test Prioritization (ATP) strategies were developed, using fuzzy analytical hierarchy process (AHP) and the weighted sum model (WSM). The empirical studies provided in this case revealed that the cost-effectiveness of regression testing can be improved by utilizing these strategies.

3. EXISTING TEST CASE PRIORITIZATION APPROACHES

3.1 Genetic Algorithm

Genetic Algorithm is a search-based optimization technique that generates optimal or near-optimal solutions to difficult problems by imitating the process of biological evolution. In order to accomplish this, it repeatedly modifies a random population of individuals, represented by chromosomes towards a better solution. During each step, it chooses individuals from the population in accordance to some fitness function of the problem under consideration. The two best fit individuals selected then act as parents to produce new off-springs. For this purpose, they undergo modifications in the form of following genetic operators:

a) **Crossover:** Crossover operator is used to introduce variation among chromosomes belonging to successive generations in such a manner that the new chromosome obtained after the crossover operation is superior over the original chromosomes. Basically, it mimics the natural selection process by taking two or more chromosomes as parents and then generating a child chromosome from them. Figure 1 below explains the process of one-point crossover, in which a random crossover point is chosen in each parent chromosome. Everything beyond that point is then swapped for obtaining off-springs.

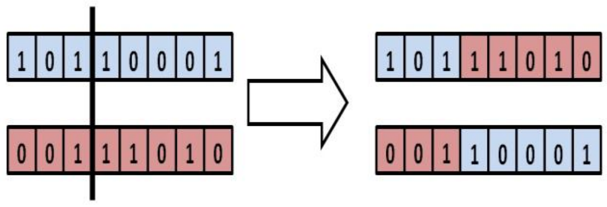

Fig 1: One-point crossover

b) **Mutation:** Mutation operator is used to insert a small tweak in the chromosome in order to produce a new solution. The solutions obtained after applying mutation can be completely different from the previous solution. Figure 2 below explains the process of bit-flip mutation, in which one or more bits are randomly selected and then flipped.

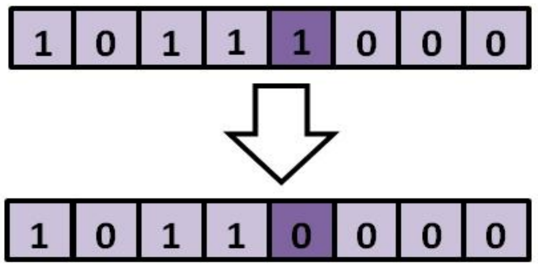

Fig 2: Bit-Flip mutation

3.2 Adaptive Approach

An adaptive approach provides solutions to prioritization problems by carrying out the processes of prioritization and execution simultaneously, in contrast to other existing prioritization approaches that perform them in isolation. This is done by calculating an initial fault detection capability (denoted as *Priority(t))* for each test case t according to its statement coverage in previous program. Then a test case t_s with the highest *Priority* is selected, which is given by the following equation:

$$Priority(t) = \sum_{S\ is\ executed\ \ by\ t} Potential(s) \qquad (1)$$

where *Potential (s)* indicates how likely a statement *s* contains faults that have been left uncovered by an existing test suite. It lies in the interval [0, 1] for each statement. Initially, every statement is assigned a Potential 1. The test case with the largest priority is then executed and its output is recorded. Based on this output i.e. pass or fail, the adaptive approach modifies the Potential of each statement *s* in accordance to the given equation:

$$Potential(s) = \begin{cases} Potential'(s), & s\ is\ executed\ by\ t' \\ Potential'(s) * q, & \begin{matrix} s\ is\ executed\ by\ t' \wedge \\ t'\ is\ passed \end{matrix} \\ Potential'(s) * p, & \begin{matrix} s\ is\ executed\ by\ t' \wedge \\ t'\ is\ failed \end{matrix} \end{cases} \qquad (2)$$

Here, *Potential' (s),* denotes the likelihood of any statement containing new faults prior to running any test case *t'*. *p* and *q* are two non-negative constants, which lie in the interval [0,1]. This process terminates when all the test cases have been prioritized as well as executed.

4. PROPOSED WORK

Test Case Prioritization using Genetic Algorithm yields excellent results. This is so because it uses the techniques inspired by the natural selection for generating the solutions. But the main drawback with Genetic Algorithm is that it consumes too much time to complete this activity. The reason behind this is the isolation of prioritization and execution processes i.e. test cases are executed only after they have been prioritized. On the contrary, an adaptive approach carries out both the processes side by side. Since both processes occur simultaneously, the time expenses are minimized to a great deal. But the problem with this approach is that it does not schedule the order of all the test cases contained in the test suite. It only prioritizes those test cases which have attained some amount of statement coverage in the past. On the other hand, the test cases which have been unable to achieve statement coverage are left non-prioritized, which implies that 100% statement coverage has not yet been achieved. As a result, a hybrid approach has been proposed in this paper, which is a combination of Genetic Algorithm and Adaptive approach. This approach works by initially employing the adaptive approach for the prioritization of those test cases which have achieved statement coverage on the previous program. Further, the test cases with no statement coverage i.e. leftover test cases are prioritized using Genetic Algorithm. This is done with the help of four operations: parent selection, crossover, mutation and duplicate elimination. In this way, the hybrid approach succeeds in overcoming the limitations of both approaches. Thus apart from saving time, the proposed approach also achieves nearly 100% statement coverage.

The methodology of the proposed approach consists of the following steps:

1. Collect different test cases from Apache Open source by interfacing it in Eclipse and testing with JUnit test toolkit.
2. Collect execution information of those test cases on the previous program.
3. Use adaptive approach for calculating the initial fault detection capability of each test case on the basis of execution information obtained.
4. Select the test case with the largest priority.
5. Run the selected test case and record its output value (passed or failed).
6. Based on its output, use adaptive approach for modifying the priority of unselected test cases and selecting the test case with largest modified priority.
7. Repeat the above process until all the test cases within the test suite are prioritized.
8. If there are any test cases which are left non-prioritized, take those for initializing the population of Genetic algorithm.
9. Apply parent selection, crossover and mutation operation on the initialized population and prioritize the test cases.
10. Calculate the execution time and APSC values for the proposed hybrid approach.

5. EXPERIMENTAL EVALUATION

For performance evaluation of the proposed hybrid approach, 100 test cases have been fetched from the Apache Open Source by interfacing it in Eclipse and then using JUnit Test Toolkit for its testing. The proposed approach has been implemented on this dataset. For the purpose of demonstrating its superiority over existing prioritization approaches, Genetic algorithm has been chosen and implemented on the same dataset. After implementing both the approaches, the performance of each has been evaluated on the basis of two factors: Execution Time and Average Percentage of Statement Coverage (APSC) values. APSC metric can be defined as the degree to which a prioritized test suite covers the statements and is given by the following equation:

$$APSC = 1 - \frac{TS_1 + TS_2 + \cdots + TS_m}{nm} + \frac{1}{2n} \qquad (3)$$

where

TS_i denotes the id of first test case that first covers the statement i in the execution sequence.

m denotes the number of statements.

n denotes the number of test cases.

The first set of each of these values for both the approaches has been obtained by varying the number of test cases of the dataset as shown by Table 1. For this purpose, five different subsets of the original dataset have been created. These contain 20, 40, 60, 80 and 100 test cases respectively. Figures 3 and 4 represent the comparison graphs of APSC and execution time values respectively for both the approaches. It can be clearly seen from these graphs that the proposed hybrid approach outperforms Genetic Algorithm in terms of statement coverage. Apart from this, it also cuts down time expenses to a great extent.

Table 1 Set Of Values Obtained By Varying The Number Of Test Cases

No. of Test Cases	APSC values (in %)		Execution Time values (in ms)	
	APSC (GA)	APSC (HY)	Time (GA)	Time (HY)
20	98.89	100.15	28484	19026
40	98.83	99.73	77285	40220
60	96.5	99.55	123043	73675
80	96.95	99.74	227599	65910
100	95.65	99.61	237198	146052

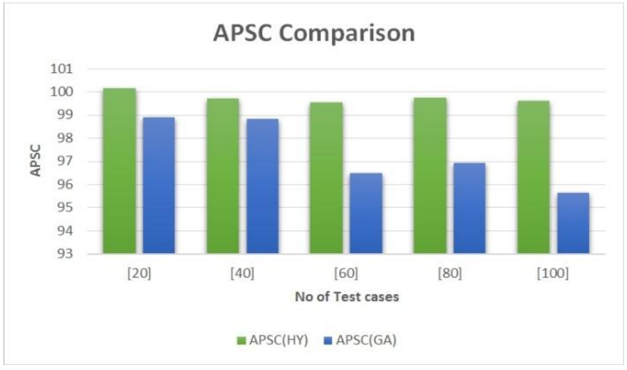

Fig 3: Graph illustrating the APSC values for Genetic Algorithm and the proposed hybrid approach obtained by varying the number of test cases.

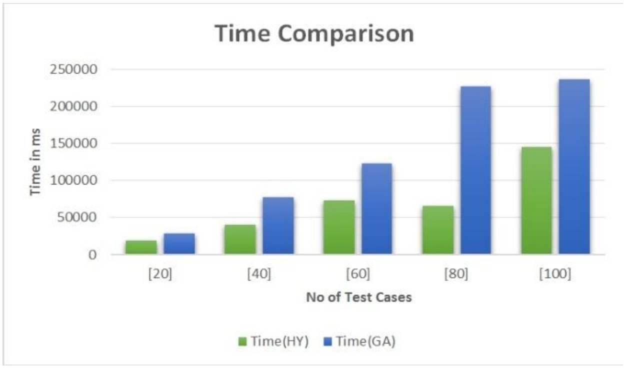

Fig 4: Graph illustrating the execution time values for Genetic Algorithm and the proposed hybrid approach obtained by varying the number of test cases.

The next set of values has been obtained by running both the approaches for different number of generations, as given by Table 2. This has been achieved by setting the number of generations to [2], [3], [4] and [5]. Figures 5 and 6 represent the comparison bar graphs of APSC and execution time values respectively for both the approaches, whereas figures 7 and 8 show the comparison line graphs for the same. From both the graphs, it is evident that the proposed approach performs better by maximizing the statement coverage up to 4 %. Moreover, a significant difference can be observed in case of execution time also.

Table 2 Set Of Values Obtained By Varying The Number Of Generations

No. of Generations	APSC values (in %)		Execution Time values (in ms)	
	APSC (GA)	APSC (HY)	Time (GA)	Time (HY)
[2]	97.53	99.6	289582	98623
[3]	95.91	99.51	411032	113702
[4]	96.12	99.6	417585	204130
[5]	95.51	99.56	468275	255237

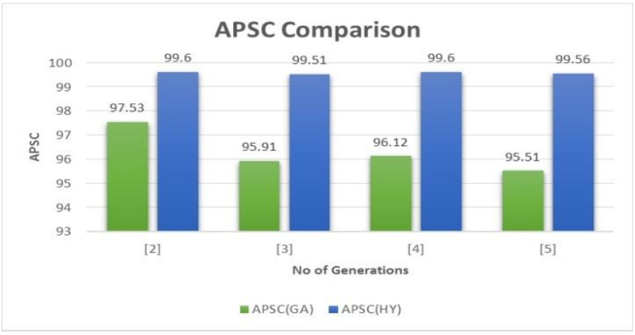

Fig 5: Graph illustrating the APSC values for Genetic Algorithm and the proposed hybrid approach obtained by varying the number of generations.

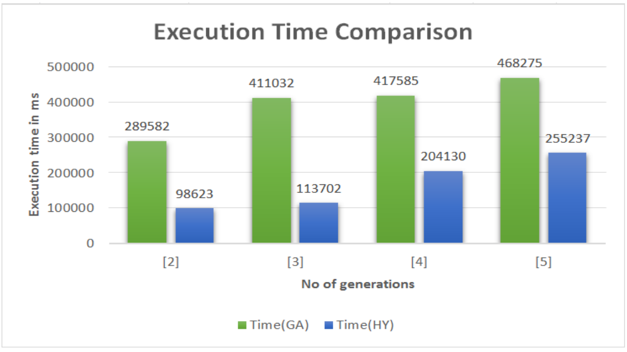

Fig 6: Graph illustrating the execution time values for Genetic Algorithm and the proposed hybrid approach obtained by varying the number of generations.

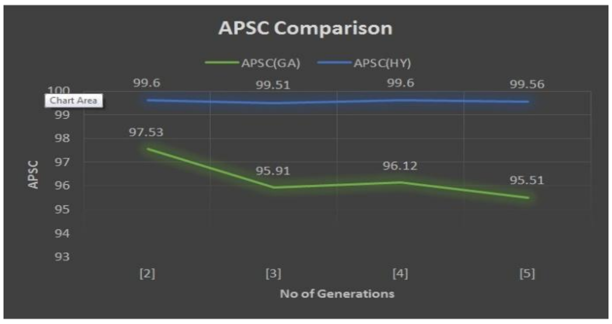

Fig 7: Graph illustrating the APSC values for Genetic Algorithm and the proposed hybrid approach obtained by varying the number of generations.

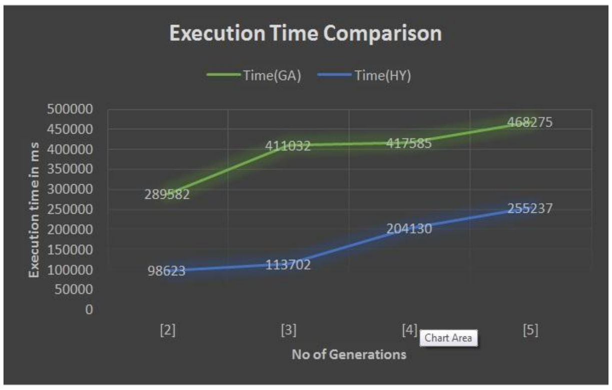

Fig 8: Graph illustrating the execution time values for Genetic Algorithm and the proposed hybrid approach obtained by varying the number of generations.

6. CONCLUSION AND FUTURE SCOPE

In this paper, a hybrid approach to test case prioritization has been presented with a view to combat the issues involved in regression testing. A combination of adaptive approach and Genetic Algorithm, the proposed approach firstly utilizes the adaptive approach for the prioritization of test cases. It works by selecting a test case with the largest priority. Then it runs that test case and records its output. On the basis of this output and the execution history of next unselected test case, it prioritizes the next test case. This process terminates when all the test cases that cover code statements have been prioritized and executed. As far as test cases with zero statement coverage are concerned, Genetic Algorithm prioritizes them using four operations: parent selection, crossover, mutation and duplicate elimination. The performance of the proposed approach when further compared with Genetic Algorithm, reveals promising results in terms of APSC values and Execution times.

The future work on this paper will focus on exploring the effectiveness of the proposed technique to a greater extent, by means of some more parameters.

7. ACKNOWLEDGEMENT

I am grateful to Ms. Harpreet K. Bajaj, Head of Department, CSE at DAVIET Jalandhar. Without her guidance and support, I would not have been able to complete this paper.

8. REFERENCES

[1] Y. Li, N J. Wahl. An Overview of Regression Testing. ACM SIGSOFT Software Engineering Notes, 25(1), 69-73, January 1999.

[2] S. Yoo, M. Harman. Regression testing minimization, selection and prioritization: a survey. Software Testing, Verification and Reliability, 22(2), 67-120, March 2012.

[3] YC Huang, CY Huang, JR Chang. Design and Analysis of Cost-Cognizant Test Case Prioritization Using Genetic Algorithm with Test History. In: Proceedings of 34th IEEE Annual Computer Software and Applications Conference, 413-418, July 2010.

[4] S. Sabharwal, R. Sibal, C. Sharma. A Genetic Algorithm based Approach for Prioritization of Test Case Scenarios in static testing. In: Proceedings of International Conference on Computers and Communication Technology (ICCCT), 304-309, September 2011.

[5] S. Mahajan, S.D. Joshi, V. Khanaa. Component-Based Software System Test Case Prioritization with Genetic Algorithm Decoding Technique Using Java Platform. In: Proceedings of IEEE International Conference on Computing Communication Control and Automation, (ICCUBEA), 847-851, February 2015.

[6] L. Ramingwong, P. Konsaard. Total Coverage Based Regression Test Case Prioritization using Genetic Algorithm. In: Proceedings of 12th IEEE International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), 1-6 , June 2015.

[7] L. Ramingwong, P. Konsaard. Using Artificial Bee Colony for Code Coverage based Test Suite Prioritization. In: Proceedings of IEEE 2nd International Conference on Information, Science and Security, (ICISS), 1-4, December 2015.

[8] Y. Singh, A. Kaur, B. Suri. Test Case Prioritization using Ant Colony Optimization. ACM SIGSOFT Software Engineering Notes, 35(4), 1-7, July 2010.

[9] K. Solanki, Y. Singh, S. Dalal. Test Case Prioritization: An Approach Based on Modified Ant Colony Optimization (m-ACO). In: Proceedings of IEEE International Conference on Computer, Communication and Control (ICCCC), 1-6, September 2015.

[10] M. Tyagi, S. Malhotra. Test Case Prioritization using Multi Objective Particle Swarm Optimizer. In: Proceedings of IEEE International Conference on Signal Propagation and Computer Technology (ICSPCT), 390-395, July 2014.

[11] T. Noguchi, H. Washizaki. et al. History-Based Test Case Prioritization for Black Box Testing using Ant Colony Optimization. In: Proceedings of 8th IEEE International Conference on Software Testing, Verification and Validation (ICST), 1-2, April 2015.

[12] A.Gupta, N. Mishra, A. Tripathi, et al. An Improved History- Based Test Prioritization Technique Using Code Coverage. Advanced Computer and Communication Engineering Technology, 315, 437-448, 2015.

[13] Md. J. Arafeen and H. Do. Adaptive Regression Testing Strategy: An Empirical Study. In: 22nd International Symposium on Software Reliability Engineering, 130-139, November 2011.

[14] D. Hao, X. Zhao, L. Zhang. Adaptive Test-Case Prioritization Guided by Output Inspection. In: Proceedings of 37th IEEE Annual Computer Software and Applications Conference (COMPSAC), 169-179, July 2013.

[15] L. Mei, W.K. Chan, T.H. Tse, B. Jiang. Preemptive Regression Testing of Workflow-based Web Services. IEEE Trans. On Services Computing, 8 (5): 740-754, 2015.

[16] A.Schwartza, H. Do. Cost-effective regression testing through Adaptive Test Prioritization strategies. Journal of Systems and Software, 115, 61-81, May 2016.