VOL. 10, NO. 4, MARCH 2015 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences


www.arpnjournals.com

STRUCTURAL SOFTWARE TESTING: HYBRID ALGORITHM FOR OPTIMAL TEST SEQUENCE SELECTION DURING REGRESSION TESTING

J. Albert Mayan[1] and T. Ravi[2]
[1]Faculty of Computing, Sathyabama University, Chennai, India
[2]Srinivasa Institute of Engineering and Technology, Anna University, Chennai, India
E-Mail: albertmayan@gmail.com

ABSTRACT

Regression testing is one of the testing methods, which is done to assure that the changes made in fixes or other improvement changes does not affect the previously developed functionality of the software. Due to the reasons mentioned below, the regression testing posses a significant place in the software testing. Since it reduces the gaps of an application, modification has to be created and tested, test the new modifications to check that the modification did not change the any other part of the application, test coverage must be enhanced without conciliation of timelines. Thus, the test case generation and test case selection is the import role in the SUT (Software Under Test). More techniques are proposed to overcome these issues however those techniques could not be achieved complete code coverage with less time duration. In this paper proposed a hybrid algorithm which is used to resolve an optimal test case sequence selection as well as new test case generation for regression testing. The experimental results are shown our proposed approach is achieved better results rather than other approaches.

Keywords: regression testing, test-case selection, hybrid algorithm, tabu search algorithm, software testing.

1. INTRODUCTION

Software testing is a critical process but it is significant in assuring the quality of the software. In the overall improvement cost of the software nearly 50% is spent for the software testing. The developers analyzed that the software testing costs for a reasonable amount in the software project budget. Therefore software quality managers are searching for the best way to decrease the total amount and time.

Software regression testing is an important component in software development life cycle. In software regression testing modified program validation and verification process will be carried out. The quality of the software can be proved to the customer by software regression testing only.

Nowadays, the data mining approaches influences the test mechanization in the field of software regression testing. Due to the changes in the customer requirements the regression testing would be more complex. The agile programming principles suggested the smaller software development life cycle also it going to suggest fewer constraints and limitations on the execution of regression testing with bounded resources. Generally, the most of the simple methods are executed all existing test cases in a test suite, it also called retest-all method. However, the test suite is grown up always if the software is modified with additional criterion. It increases the execution cost of the test suite. This problem is forcing to the more techniques require on the regression testing in order to reduce the cost.

Many approaches are analyzed to support the regression testing process. Test case selection, test case minimization and test case prioritization are the three important steps in the regression testing. Test suite minimization is a method or technique which is used to find and eliminate the unused test cases from the test suite. Test case selection is the technique in which it selects the division of the test cases which would be utilized to test the altered parts of the software. Prior experimental analysis explains that the use of these techniques would be gainful. Regression testing technique is used to confirm the quality of the modified software. The regression test suite is naturally huge and requires a smart method to select those test cases to detect the utmost or all the faults at the starting stage. Software Testing Coverage is a significant pointer of software quality and an important part of software maintenance. It is useful in analyzing the value of testing by given data on different coverage items. Lots of research attempt made to get coverage information by either code based testing or requirement based testing, but not more effort has been made to determine and study the coverage by covering maximum number of coverage items [3] [4].Number of approaches is proposed to decrease the time and costs for the software testing process, which also contains test case prioritization techniques and test case reduction techniques. It is explained well that prioritizing and scheduling test cases are one of the most significant tasks during the software testing process[5] [6].

Although test case selection and test cases methods have great benefits for software test engineers, there are still outstanding major research issues that should be addressed. In this research paper there is two problems. They are an optimal test cases selection and abstract method testing problem. Initially, every historical based method such as genetic, Tabu search and random search algorithm selects the test cases in arbitrary order. This

process sometimes may be leads to unnecessary test cases selections.

This paper is explored with the concept of test case selection and test suite using hybrid algorithm where test cases are selected based on their suitable offspring generation for modified program and test case suite also done based on the hybrid algorithm. Test case selection and suite techniques select and prioritize and schedule the test cases in an order that attempts to maximize some objective function. For this solution by using Tabu list in the frequent access for long term and short term list.

The proposed frame work for an optimal sequence is a long and short term list. So, the hybrid algorithm is used to resolve an optimal test case for frequent used in long term list and less frequent used in short term list. The software testing problem for test cases selection is to develop a system where the developed can re-test and debug their software, with new proposed algorithm in the field of regression testing and debugging software.

2. RELATED WORK

The reliable software testing is needed to the software engineers in order to build up the High-quality systems. As per M. J. Harrold [34] the software testing ensures the quality of the software by collecting all specifications of the type of the software. The testing comprises these following actions developing the test cases, testing the modified software with previously generated test cases, after that investigating the outputs generated by those testing. Boris [36] describe that nearly fifty percent of the software development cost is utilized for testing, in case of an important software the cost of testing is very high. When there is development of a important software then the tasks performed by the software will also high, hence the quality of the software should also be high. If the engineers failed to find prominent way of testing then the cost of testing is also increased consequently.

Testing techniques are the methods which are used in the execution of the program or application in order to detect the software bugs. In other word, it can be defined as the validation or verification process of the software, which ensures whether the software satisfies the business and technical requirements as specified during its development process, hence assures the proper working of the software [30]. Software testing can be utilized at any instance of the development process. Software engineers usually accumulate the developed test suites for reusable purpose in the development process of the software. The re-usage of the test cases in the regression testing process is very significant in the software industry [34] by this process half of the software maintenance cost is reduced [36].

Boris [8], said that software testing would utilize 40-70% of the time and cost of the software development process. In order to reduce time and cost of software testing process, several approaches have been suggested. The test case prioritization techniques and test case reduction (selection) techniques are some of the techniques which are used in the suggested approaches. In order to achieve the aim of testing as soon as possible in the testing process the Elbaum *et al.* [18] proposed a technique known as selection and prioritization techniques. In their research they are discussed about rate of fault detection {an evaluation of fault detection time} which is one of the main aims of the testing process. The highest fault detection rate will be gives the fast feedback about the tested software that is also enables the fastest debugging but the minimized test case sequence based testing may be completed before reaching destination, however the sequence can be detected the maximum faults in the modified part of software with less time.

Elbaum *et al.* [22] [23], Rothermel *et al.* [20] [21] selected four heuristics for the target prioritization techniques in their literature and investigation. Their literature and investigations in experimental studies let them to inspect the two of the key scope of differences among techniques. These scopes are the uses of response and information on modifications. For effortlessness and to make easy comparison, Elbaum et.al. Rothermel et al. restricted their attention to function-coverage-based techniques. Zheng Li et al. have tested experimentally that genetic algorithms perform well for test case selection and prioritization [24].

The history-based test case selection and prioritization technique was suggested by Kim and Porter [25]. This technique uses the data from early testing cycles to choose the test cases which should be executed for the new version of the program. Since this technique does not implement ordering on test cases, this technique is not referred as “prioritization technique”. Ordering of tests is the important characteristic for the definition of prioritization. In this approach, the subset of a test suite is selected and based upon the history information the test cases should be preferred and it is predominantly described as a “regression test selection technique” by Rothermel and Harrold [26].

As mentioned above, Rothermel [38], [39] provided a remarkable model of industrial collaborators. In those situations, testers would like to regulate their test cases so that those test cases with the highest priority, according to some criterion, are run first. The test case prioritization techniques is used to prioritize and schedule the test cases in such a way that it tries to maximize several objective function. For instance, the test cases are scheduled by the software tester in order to obtain the maximum code coverage and detect all faults very quickly which is also reflects on the future historical usage [4]. Some of the remarkable feature in test case prioritization is: when the required time for the execution of all test cases in the test suite is short then the test case prioritization is a prominent method. When the required time to execute all the test cases in the test suite is more than the test case prioritization is a profitable method. Test case prioritization techniques provide a way to schedule and run test cases, which have the highest priority in order to provide earlier detected faults. Test case prioritization

techniques offer an approach to schedule and run test cases, which have the highest priority in order to provide earlier detected faults.

3. PROPOSED WORK

Our proposed work mainly overcomes the problem of test case selection in genetic algorithm. Normally in genetic algorithm the test cases are selected randomly and perform remaining operation. But the main disadvantage is the randomly selected test cases may not be gives optimal test cases to test the modified program. So our proposed algorithm selects the test cases based on the number of frequency occurred of every test case, the maximum frequency test cases are selected for testing the modified software program and then again these test cases are represented as long-term list. These test cases cover more number of lines and finds maximum faults within less time.

3.1 Define optimal test case selection problem

For a program (or a function) that Genetic algorithm takes finite set of input parameters, each of which has a specified input domain, testing all combinations of parameter values is referred to as exhaustive testing. The number of parameter combinations can be very large, which makes exhaustive testing not applicable in most practical cases.

In early, more number of users used Genetic algorithm concepts for test case generation, selection, cross over, and mutation. But the main disadvantage of genetic algorithm is random test case selection in every testing process. Normally in test case selection the genetic algorithm selects test cases randomly for software under test. So the selected test may not produce high fault detection rate and also could not be covered maximum codes within less time.

4. OPTIMAL TEST CASE SELECTION: HYBRID ALGORITHM

This concept of test case selection on the normal genetic algorithm, sometime it generated unwanted test sequence in random selection operation. To avoid this issue, the hybrid technique is used. The hybrid technique is similar as genetic algorithm but the mutation operation done by long term test cases and short term test cases where two tables are maintained for short term and long term offspring. The following technique is utilized to generate optimized test case sequence.

4.1 Generate population

Based on the assumed program module, the groups of test cases are orderly or arbitrarily generated which is known as test case generation in the hybrid approach. Then the syntax is assigned for all test cases to test the modified program module. First choose order/random number of parameters and combine the input parameters as randomly, and then finally assign the syntax to all generated test cases. According to the syntax and the input parameter is applied in the testing program. At last the test suite has been generated.

4.2 Selection criteria

A program P is considered as the initial version program where a test suite is applied on this program then there are two types of list is generated such as the more frequently utilized test cases as long term list and less frequently utilized test cases as short term list. After that, a program P is considered its test suite and its modified program P'. Once the program P is modified to P' then the source code is affects with these modified portions. So, the all parts of the modified program must be test with less selected test cases. Thus, the proposed hybrid approach selects the minimum set of test cases based on their high code coverage, less execution time and high fault detection rate. The selected test cases are applied on the modified program with any one of the following reason: 1) At least one time execute the modified source code lines with selected test case 2) After deleted the history of executed test cases or unnecessary test cases then execute the source code. Thus, the hybrid approach can be operated with following information 3) A program P and its modified Program P' 4) A test suite TS with number of test cases tc1, tc2… tc_n 5) history of every test case execution i.e., code coverage, execution time, fault detection rate.

Pseudo code: Hybrid algorithm

1. Initialize TC (Test case) sets
2. Generate functions
3. for i= 1; i< size of TC ; increment i;
4. count every repeated test cases
5. end for
6. for j=1; j<total test cases; increment j;
7. if total frequency of test case[j]> threshold then
8. do
9. long-term list=test case[j]
10. else
11. do
12. short-term list=test case[j]
13. end if
14. end for
15. for k=1; k< long-term list AND short term list; increment k;
16. if (Code Coverage< MAX)
17. do
18. apply long term test case[k]
19. OR else (long-term list=ϕ AND Code Coverage<max)
20. do
21. apply short term test case[k]
22. end for
23. if(Code Coverage<max) AND long-term list & s hort term list=ϕ
24. while (Code Coverage<MAX OR iteration<0)
25. select random test cases (Population Size) ∈ long-term, short term and Test case set
26. apply Population test cases
27. if(Code Coverage<MAX)

28. select n best solutions
29. select two parent p1 and p2 test cases ∈ Population according the selection criteria
30. Generate offspring from p1 and p2
31. apply cross-over probability to generated offspring1
32. child test case=Mutate generated offspring
33. n solutions= n solutions∪ child test case
34. Population = Population ∪ n solutions
35. end while
36. end if
37. Return Optimized Test cases

4.3 Offspring selection

In this paper, the test case selection process is based on the hybrid approach. This approach is using the core operation of genetic algorithm where it only differs from the following process 1) population selection and 2) new child test case generation. The population set is selected based on the execution history such as code coverage, execution time and fault detection. The line 1 to 25 shows the historical based test case selection process. The offspring is generated using two parent test case's input parameters. The parents are selected based on the utilization in the testing process.

4.4 Cross over probability

The hybrid approach provides the best solutions for test case optimization problem. The domain based input parameters that are used to achieve our aim of the required testing results using with proposed hybrid approach. Consequently, the additional target is achieved by utilizing the new inputs that are produces with previous inputs when they joined mutually. The crossover probability is used here to generate the suitable new test case inputs. The crossover probability is the fitness function of the population test case. The development is accomplished on basis of the two primaries operate known as crossover and mutation. The old knowledge of the parent population is used to produce the better performing new solutions. The finest solutions are reused while the worst solutions are avoided. The inhabitants consist of the arbitrarily generated group of individuals. The selection of the initial generation is the main chore that considerably impact effect on the presentation of the next generation. The single test case in the input parameter binary stream is referred to a chromosome. The chromosomes are altered with the help of the crossover and mutation operators. The fitness of each entity is measured to evaluate the performance of all entities. The fitness value of the individual that present close to the optimum solutions will be high. By utilizing the process of mutation and crossover new members are formed. The crossover and mutation are generally used operators. It is operated in the binary form of the input parameters. The lines 26 to 37 is explored the functions of cross over probability and mutation.

4.5 Mutation

The mutation probability is utilized in this proposed system. The mutation is the process in which the test case inputs are muted inside a population (For example we may consider the test cases with execution history). Then the realistic execution of the hybrid algorithm specifies about such things about the code that would be tested. By using this algorithm a parameter can locate the line code which is covered by the test cases, the test case Id and the statement numbers enclosed by particular software. This algorithm is very supportive because with this information the test case which covers the majority of the modified lines of code can be identified by comparing with the number of modified lines. All test cases should be linked with inherent properties like the code functions they pass through the development code, the complexity of the tested code. By this the test case which covers the lines but not the modified lines of code can be identified. The hybrid algorithm is the greatest search problem that stored long term and short term test cases hence this algorithm is used.

Then the modified software program is given as input for GA to test the program. Now the most frequent test cases are in collection of test cases is considered as long term test cases and the less frequent test cases are considered as short term selection. Then the selected long term test cases are considered for testing process. The proposed hybrid algorithm reduces the time for selecting the test cases, improves the rate of efficiency, reduces the memory space, increases the speed of execution and reduces the cost of test case generation.

5. RESULTS AND DISCUSSIONS

A test suite contains collections of test cases to identify the maximum number of faults in software program under test. The result explains about the process as well as benefit of our proposed system. Initially the proposed hybrid Algorithm takes software program as input and select test cases occurred more frequently in collection of test cases and these test cases are considered as long term test cases, and the less frequent occurred test cases are considered as short term test cases. After test case selection the long term test cases, these test cases are applied to software program to cover more number of faults, then cross over and mutation process are applied. The main advantage of our proposed algorithm is improve the fault detection rate by using fewer test cases, obtains high efficiency test cases and reduces the memory space.

5.1 Experimental setup

In these experiments, we implemented and evaluated the proposed framework in Java platform on a Microsoft Windows XP Professional with Intel Pentium IV Dual-Core Processor 1.83 GHz CPU and 2 GB RAM. A bank application functions are developed in OOPs program. Also it includes number of program versions. The every program versions contains more than 500 lines of codes. The every program versions are tested with below shown fragmented test case Table.

Table-1. A fragmentation of test suite.

S. No.	Test case	Input	Syntax
1	TC1	Nmewgdij/ cbfbbbb/ Female/ gahfvhhm/ knytutbitj/ empty, Ewej	CreateAccount, checkMinimumBalance
2	TC2	Ypsm/ wajoafwllqv,Male ,Cvhhloches/ 675/ Male/ vjjiwsbujylno/ fr/ nf, Bghfccgdgh,Wmv tocgtmh,Lbs	checkBalance, checkGender, CreateAccount, checkAge, checkMinimumBalance, checkAddress
3	TC3	Ndql/ 37/ Male/ fniuyvwoxu twienonvoa/ mpgvg, Odr/ qr/ qxy, ,Lugfrpubxhtlb/ lt/ uawu	checkAmmount, CreateAccount, Deposite, Widthdrwel

Table-1 shows the fragmented test suite and it includes amount of test cases. Every test cases are includes the set of inputs and syntax to test the modified version program. These test cases are tested on given source code in sequence or arbitrary order.

Table-2. Test cases frequency.

Test case ID	Frequency
TC1	3
TC2	8
TC3	2
TC4	4
TC5	7

Table-3. Fragmentation of Short-term list and Long-term list.

S. No.	Short-term list		Long-term list	
	Test Cases Id	Frequency	Test Cases Id	Frequency
1	TC1	3	TC2	8
2	TC3	2	TC5	7
3	TC4	4		

The Table-2 and Table-3 show the short term list and long term list is generated based on the frequency of test cases i.e., which is testing the initial/modified program. The short term list contains less frequently occurred test cases and long term list contains more frequently occurred test cases.

Table-3. Fragmentation of Test cases code coverage, fault detection and their execution time.

S. No.	Test cases Id	Covered Lines	Execution time (Milliseconds)	Detected faults
1	TC1	15	234	4
2	TC2	20	150	3

The Table-3 shows detected faults and their execution time. The test case TC1 covered lines is 15, execution time is 234 and detected fault rate is 4. Also TC2 the covered lines in the software program is 20, execution time is 150 milliseconds and the total detected fault is 3.

5.2 Comparison of hybrid algorithm with genetic and random search

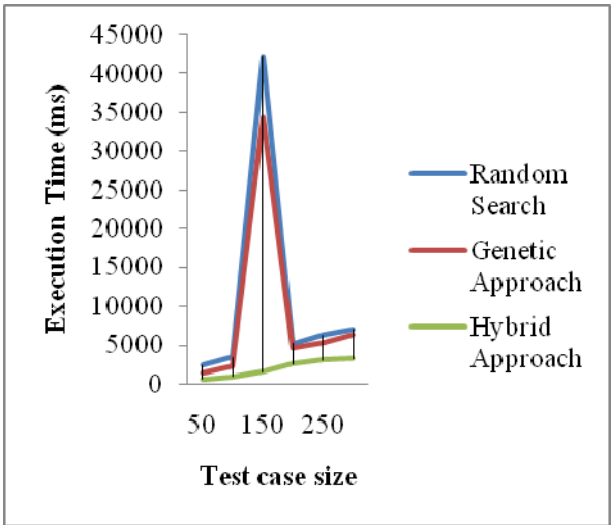

Figure-1. Comparison of test cases and execution time with different approaches.

The Figure-1 illustrates the comparison of number of test cases and execution time. In this figure, initially the execution time increases as per number of test cases. After particular test cases the difference of time will reduce slowly but the proposed algorithm execution time increases in linear with less time.

Figure-2. Comparison of test case size vs fault detection rate.

This Figure-2 expresses the comparison of test case size and fault detection rate of proposed algorithm. This graph explains that the test case detects more number of faults with fewer test cases also if the test size increased the fault detection rate also increased because of the mutation function generate suitable child test cases with less time.

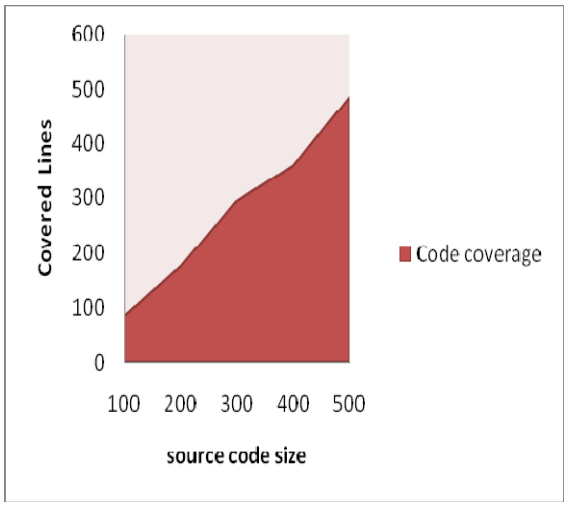

Figure-3. Comparison of source code Vs code coverage.

Figure-3 expresses the comparison of source code and code coverage. i.e., the proposed hybrid algorithm covers more number of source codes lines within less time.

5.3 Implementation results

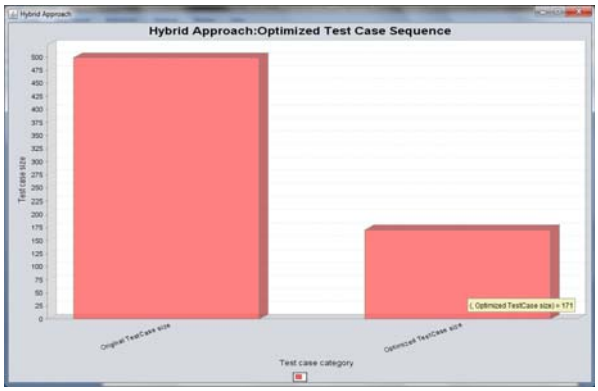

Figure-4. Hybrid approach's optimized test suite.

The Figure-4 shows the optimized test case sequence that is generated by proposed hybrid algorithm. The whole test suite size is 500 but the 150 test cases only used to test the modified program.

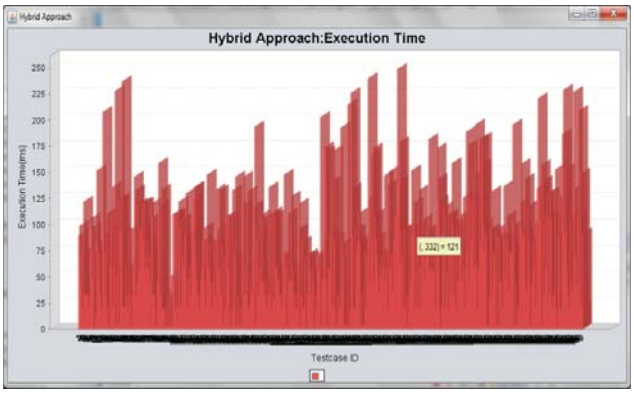

Figure-5. Test cases execution time in hybrid approach.

The Figure-5 shows execution time of every optimized test case in the hybrid approach. These test cases are taken very less time to test the modified programs.

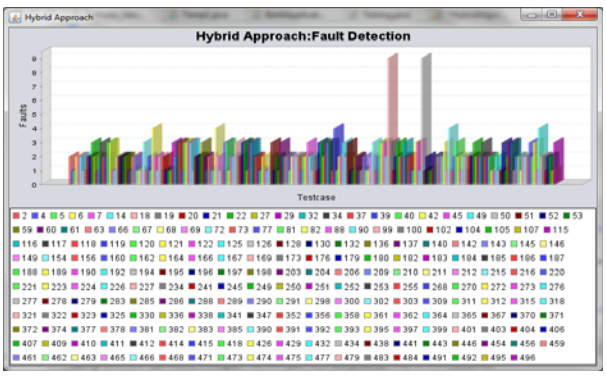

Figure-6. Hybrid algorithm: fault detection by every test cases.

The Figure-6 shows the fault detection of every test case in the hybrid approach. It detects almost all faults in the modified program. The fault may be misplaced conditions or missing conditions based on the client's specification

6. CONCLUSIONS

In this research paper, the proposed Hybrid Algorithm is used to create Long term and short term test cases. Long term test cases contain more frequently occurred test cases and the short term contains less frequently occurred test cases. The more frequently occurred test cases are used to test the software at first iteration. After program modification process this proposed algorithm again generates the long term test cases to test the modified program instead of generating new test cases. So this again generated long term test cases are more effective in finding faults in software program. So our proposed algorithm reduces the time for test case generation improves the effectiveness of testing. In our future work, we have planned to add this method with other testing frameworks to improve the testing performance.

REFERENCES

[1] Z. Li., M. Harman. and R. M. Hierons. 2010. Search Algorithms for Regression Test Case Prioritization. IEEE Trans. on Software Engineering. Vol. 33, no. 4.

[2] G. Rothermel., R. Untch., C. Chu. and M. Harrold. 2009. Test Case Prioritization: An Empirical Study. Int. Conf. on Software Maintenance. Oxford, UK, pp. 179 - 188.

[3] K. R. Walcott. and M. L. Soffa. 2011. Time Aware Test Suite Prioritization. ISSTA'11, Portland, Maine, USA.

[4] M. Fayoumi., P. Mahanti. and S. Banerjee. 2010. OptiTest: "Optimizing Test Case Using Hybrid Intelligence" World Congress on Engineering.

[5] S. Mirarab. and L. Tahvildari. 2010. An Empirical Study on Bayesian Network-based Approach for Test Case Prioritization. Int. Conf. on Software Testing, Verification, and Validation.

[6] A. G. Sanchez. 2010. Prioritizing Tests for Software Fault Localization. 10th Int. Conf. on Quality Software.

[7] 2010. Journal of Theoretical and Applied Information Technology, JATIT & LLS. All rights reserved, www.jatit.org, Test Case Prioritization Techniques, SIRIPONG ROONGRUANGSUWAN, 2JIRAPUN DAENGDEJ, Autonomous System Research Laboratory, Science and Technology, Assumption University, Thailand.

[8] Arup Abhinna Acharya., Durga Prasad Mohapatra. and Namita Panda. 2010. Model Based Test Case Prioritization for Testing Component Dependency in CBSD Using UML Sequence Diagram. (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 1, No. 6, December.

[9] H. Leung. and L. White. 2009. Insights into regression testing. In Proceedings of the International Conference on Software Maintenance. Miami, Florida, U.S.A., pages 60-69, October.

[10] Jefferson Offutt, Jie Pan and Jeffery M. Voas. 2008. Procedures for Reducing the Size of Coveragebased Test Sets.

[11] Dennis Jeffrey. and Neelam Gupta. 2006. Test Case Prioritization Using Relevant Slices. In: Proceedings of the 30th Annual International Computer Software and Applications Conference, Volume 01, pages 411-420.

[12] S. Elbaum., D. Gable. and G. Rothermel. 2011. Understanding and measuring the sources of variation in the priori- tization of regression test suites. In Proceedings of the Seventh International Software Metrics Symposium. Institute of Electrical and Electronics Engineers. Inc., April.

[13] G. Rothermel., R. Untch., C. Chu. and M. Harrold. 2009. Test case prioritization. An empirical study. In: Proceedings of the International Conference on Software Maintenance. Pages 179-188.

[14] G. Rothermel., R. Untch., C. Chu. and M. Harrold. 2011. Test case prioritization. IEEE Transactions on Software Engineering. 27(10):929-948, October.

[15] S. Elbaum., A. Malishevsky. and G. Rothermel. 2011. Incorporating varying test costs and fault severities into test case prioritization. In: International Conference on Software Engineering. pages 329-338, May.

[16] S. Elbaum., A. Malishevsky and G. Rothermel. 2012. Test case prioritization: A family of empirical studies. IEEE Transactions of Software Engineering, 28(2):159-182, February.

[17] G. Rothermel., R. H. Untch., C. Chu. and M. J. Harrold. 2010. Prioritizing test cases for regression testing. IEEE Transactions of Software Engineering. 27(10): 929– 948.

[18] J.-M. Kim. and A. Porter. 2012. A history-based test prioritization technique for regression testing in resource constrained environments. In: Proceedings of the International Conference on Software Engineering, May.

[19]G. Rothermel. and M. Harrold. 2010. Analyzing regression test selection techniques. IEEE Transactions on Software Engineering, 22(8):529-551, August.

[20] Journal of Theoretical and Applied Information Technology. JATIT & LLS. All rights reserved, www.jatit.org, Test Case Prioritization Techniques, Siripong Roongruangsuwan, 2jirapun Daengdej, Autonomous System Research Laboratory, Science and Technology. Assumption University, Thailand. 2010.

[21] Z. Li., M. Harman and R.M. Hierons. 2011. Search Algorithms for Regression Test Case Prioritization. IEEE Transactions on Software Engineering. 33(4):225-237, April.

[22] S. Elbaum., G. Rothermel., S. Kanduri. and A. G. Malishevsky. 2011. Selecting a Cost-Effective Test Case Prioritization Technique. Software Quality Journal. 12(3): 185-210.

[23] Gregg Rothermel. and Mary Jean Harrold. 2010. A Framework for Evaluating Regression Test Selection Techniques.

[24] Jefferson Offutt., Jie Pan. and Jeffery M. Voas. 2009. Procedures for Reducing the Size of Coverage based Test Sets.

[25] B. Korel. and J. Laski. 2011. Algorithmic software fault localization. Annual Hawaii International Conference on System Sciences. pages 246–252.

[26] Cem Kaner. 2009. Exploratory Testing, Florida Institute of Technology, Quality Assurance Institute Worldwide Annual Software Testing Conference. Orlando, FL.