International Journal on Emerging Technologies **10**(2): 221-225(2019)

ISSN No. (Print) : 0975-8364
ISSN No. (Online) : 2249-3255

An Integrated Methodology for Testing Source Code by Using Multi Constraint Reduction, Test Suite Prioritization and Prioritized Parallelization

***Pradeep Udupa*[1] *and A. Rijuvana Begum*[2]**
[1]*Research Scholar, Department of Computer Science and Engineering, Prist University Thanjavur Vallam, India.*
[2]*Associate Professor, Department of Electronics and Communication Engineering, Prist University Thanjavur Vallam, India.*

(Corresponding author: Pradeep Udupa)
(Received 25 April 2019, Revised 12 June 2019 Accepted 02 July 2019)
(Published by Research Trend, Website: www.researchtrend.net)

ABSTRACT: Software Testing is the method of assessing a system or its components. It is used to execute a system in order to rectify any breach, bugs, or missing requirements in conflicting to the actual requirements and excluding the delusion. It requires validating an attribute to test that whether it produces anticipated and demanded outputs. So testing software plays major role in software development, but as no of test cases increases time, cost, effort, faults, complexity increases, so we use proposed technique in which we used apfd, test suit reduction, prioritization. Also in test case ordering to extend performance and an algorithm is built to optimize the overall testing coherence and to lessen the implementation time by diminishing number of test cases, performing prioritization and bug detection. In further, prioritized parallelization is done to prolong performance.

Keywords: Test suit depletion, Prioritization, Parallelization, Fault detection, Optimization.

I. INTRODUCTION

A. Software Testing
Software Testing is the technique of validating, verifying and correcting the error. It is used to ensure the software quality and completeness, here the goal is to minimize total test runs [1], because as the number of test case increases it takes immense time to test, therefore here we first try to minimize test cases then prioritize them and finally optimize them.

B. Black Box Test
It is the process of checking and testing that a software program or application or product Meets the business and technical requirements that are guided it's design and development, here whether software behaves properly or not is invisible to the testing team as in [7].

C. White Box Test
In this case process of checking whether incorporated function works properly or not, implementation details are appropriate or not, whether software behaves properly or not are invisible to the testing team. Test cases are used to define required and expected output and it is used for testing against user requirements and against specific criteria's to be satisfied.

D. Check Lists
Check lists is a condition or set of conditions for evaluating a particular feature of a software product to determine its compliance with the business requirements. A test case has pre-condition, input values and expected outcome, it is used to detect the correct behaviour or operations and characteristics of an application and an expected outcome or an expected result, it is used for rectifying whether we can able to obtain our demanded output as per our requirements which is stated at the beginning as expected result.

E. Path Validation
It is Used to test every possible path, It can be used when number of paths are more and testing more number of paths are complex and time consuming, it will be helpful in checking different criteria's to be satisfied by the given program, it includes identifying different possible path, and testing different possible path.

F. Control Flow Testing
It includes testing each possible path, it can be used when number of paths are supplemental and testing alternative paths are composite and time consuming, it will be helpful in checking alternative criteria's to be contented by the given program it includes identifying alternative possible path.

G. Independent Paths
It is a separate path in the program which is used to test specific condition and different benchmark to be satisfied by the program and engender test cases for each and every derived unique path by making particular path to pass the required condition.

H. Why to Attenuate Comprehensive Test Suits?

1. Extensive test cases leads more convolution as in [9].
2. Larger the test cases more will be the probable number of errors.
3. Error tracing is to be executed.
4. Vast no of testers are demanded.
5. It will take immense time. It will expand The Overall outlay as in [2].

I. Proposed Algorithm (How)

In proposed research we attempt to lessen number of test cases by discovering (how) min, max, and constant values in the whole test cases though locating no of test paths, here we use following steps to lesson no of test cases.

1. Spot criteria's from begin to end nodes. A condition can be (>, >=, <, <=, ==! =)
2. Locate the variables with peak and smallest values in the path, then the large variable is given high value and small variable is given low value.
3. Dwindle time required to run test cases
4. To prioritize the test cases.
5. To contract total effort
6. To eschew critical software failure
7. To rectify maximum no of errors.
8. To discover critical errors as early as Possible.

II. LITERATURE REVIEW

There were numerous papers which investigated the role of reducing test cases. Here are the several test case diminishing techniques that are explored previously.

A. Curtailment Based Test Input Procreation

DeMillo and Offutt introduced procedure for test input procreation that uses path inspecting, symbolic verification, and lessen number of test cases based on criteria [12].

B. Dynamic Domain Reduction (Ddr)

Offutt *et al.,* have invented get split algorithm which cleaves domain to dwindle overall domain range and shorten test runs and overall time needed. It has accomplished vague depletion in test cases but it is less efficient and large time swallowing and is comparatively more extortionate method [13].

C. Ping-Pong Technique

This approach selects the small number of test cases by ordering differently, and it uses heuristic process which wont ensure best output, but it can give better outcome in given time by diverging the set of values of goal state and set of states of attained values and it will assure range sheath, but it is absorbs more time and it will be exorbitant technique and more effort needed [5].

D. Test Case Reduction Using Multi Constraint

Reduction Technique & Fault Detection (proposed). Numerous techniques are entitled previously in literature but in our technique we used test case diminishing approach, prioritization, fault detection and parallelization where low, high, constant variable in all path are examined by analyzing each and every individual path, here unique paths are examined by using cyclomatic complexity and later more than one test case are executed in parallel and prioritized fashion which has increased percentage of depletion in terms execution time and number of test cases by using proposed method we perform test case depletion, prioritization and then finally test suits will be executed so that it can lesson debugging effort as in [6], and then we prioritize test cases based on test case ordering, here rectification of fault apfd calculations are performed, then rate of fault detection, percentage of fault detection, risk detection analysis are performed, here several formulas are incorporated and test case ranking is assigned for different test cases then it will be executed based on test case rankings, finally collation among different prioritized test cases is done based on fault detection rate to demonstrate that our technique has optimum performance over other existing techniques, ultimately we run test cases based on priority, this influences to expose maximum number of faults and execute test cases with severe test cases first and helps in nullifying software from failure.

Table 1: Summary of review comparison.

Author	Approach	Advantage	Disadvantage
DeMilli, & Offutt 1991 [12]	Constraint based testing	It uses control- flow analysis, symbolic evaluation and reduces no of test cases based on criteria	More no of test cases Time consuming More expensive less efficient More effort no parallelization
Offutt *et al.,* 1999 [13]	D.D.R.	Achieved more depletion in test cases	More test cases compare to constraint bases comparatively more Time consuming 3.more expensive 4.more effort 5.no parallelization
Srikanth *et al.* 2005 [11]	Prioritization techniques no order, reverse order	Attempt to detect possible no of errors and contribute in performance	Less efficient compare to proposed technique in terms of performance

III. EXISTING METHODOLOGY

Ddr Technique

Presume that given domain is i1(0..30),j1(0..50), k1(0..40) here following steps followed 1. Detecting all criteria's from begin to end

2. Examine split point value for given domain and for all variable fulfilling criteria.
3. Then as per split vale we separate into two intervals.

i1=0 to 15 and 16 to 30 i2 into 10 to 30 and 31 to 50 & final interval by using splitting is i1 0 to 10 and 11 to 30 i2 31 to 50 i3 is 10 So total test cases=31*1+31*20=651.

IV. PROPOSED METHODOLOGY

In this segment first we minimize test cases by our given algorithm then we prioritize test cases by assigning rankings for test cases then we discover APFD value which will manifest that proposed technique is better than prevailing technique, then we parallelize our test cases to lessen time and cost involved. Here first we discover number of test paths then from each path we detect min, max, and constant values and will derive our diminished test cases by using steps given below then further execute them in parallel fashion. Assume that the path 1-2-4-8 is adopted and the inceptive domains taken are i1(0..30),j1(0..50),k1(0..40).

We follow following steps.

1. Identify criteria's from beginning to end nodes. I1 < j1,

j1> = k1 then
2. Identify min and max values in the path and allot to min and max variable.
3. Determine fixed values. 'k1' fixed value obtained on node2 is allotted to variable k1.
4. Determine fixed values. 'k1' fixed value obtained on node2 is allotted to variable k1,then make use of obtained range to derive reduced test cases for all unique paths as given in Fig. 1.

Table 2: Generated Domains for Different Variables.

Test table			
Variable I 1	**Variable J1**	**Variable K1**	**Test cases/ path**
0 to 30	50	10	T1 /p1
0 to 9	10 to 50	10	T2/p2
10 to 30	0 to 30	20	T3/p3
30	0 to 50	20	T4/p4

V. RESULT EVALUATION

In this section we discriminate proposed methodology with the existing method Get Split with respect to produced total checklists, total depletion in test cases, comprehensive bugging time and fault detection rate in proposed technique for path1 we need only 31*1*1=31 is test cases.

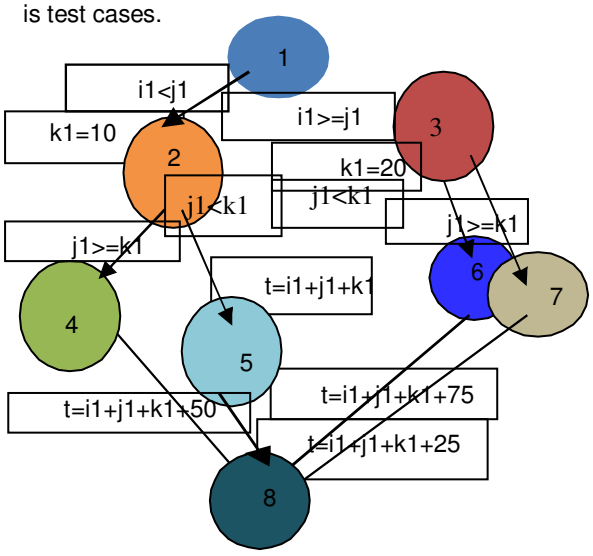

Fig. 1. Control Flow Graph.

Table 3: Number of faults exposed and time required for each Test Cases.

Test Cases/Errors	T'1	T'2	T'3	T'4	No	rev	prop
Fault1'/F1'				*'	4'	1'	1'
Fault2'				*'	4'	1'	1'
Fault3'	*'		*'	*'	1'	4'	1'
Fault4'		*'	*'	*'	2'	3'.	1'
Fault5'		*'			2'	3'	4'
Fau.lt6'			*'		3'	2'	3'
No of faults	1	2	3	4			
time	1.5	5	7	9			
severity	4	6	8	10	16	14	11

Here adopted domain is i1(0..30),j1(0..50),k1(0..40) here F1 is fault value inputted less than minimum range in which values for i1=-1 and F2 is fault value entered higher than maximum range and in which value for i1=31,and F3 is fault value inputted less than minimum range where value for j1=-4 and F4->fault value entered higher than maximum range and value for j1=51 where F5->fault value inputted less than minimum range and value for k1=-5 where F6->fault value given is higher than maximum range and value for k1=41, then for faults and severity given which is mentioned in Table 3. We then calculate rate of fault, percentage of fault and risk detection analysis, by using the formula given below. then values of rft, apfd and rda are added.
RFT=Nj/TIMEj*10 as in Table 4.
PFD=NJ/total no of faults*10 RDA=NJ*SJ/TJ
TCR=RFD+PFD+RDA as in Table 5.

Table 4: Test cases With Procured Fault Rate, Pfd, Rda and Test Case Rank.

Test cases	RFT	PFD	RDA	TCR
T1	10	3.33	4	17.33
T2	6.66	3.33	4	13.99
T3	10	5	8	23
T4	10	6.66	10	26.66

Table 5. Ranking Values For Test Cases.

Test Cases	TCR=RFD+PFD+RDA
T1	17.33
T2	13.99
T3	23
T4	26.66

Table 6: Assessment of Proposed Prioritization With Different Techniques.

No Order	Reverse Order	Proposed Order
T1	T4	T4
T2	T3	T3
T3	T2	T1
T4	T1	T2

Table 7: Calculated Apfd Values.

Prioritization Techniques	APFD%
NO ORDER	46
REVERCE ORDER	54
PROPOSED ORDER	58

Application 1: to assess continuous improvement of student, here if students first internal mark is greater than 2nd internal and 3rd internal 25 marks/credits
Added if 2nd internal is greater than first internal and 3rd internal 50 marks added and if 3rd internal is greater than first internal and 2nd internal 75 marks added as in Table 9.
Application 2: promoting banking/finance business providing added credit points for increments in deposits/loan above 5k or certain limit settled by company and can enchased as cash (Table 10).
Application 3: promoting business by providing credit Increments for increments in purchase amount for purchase amount above 500 rupees as in Table 11.

Table 8: Application Based Research Comparison of Proposed Method (Multi Constraint Based Reduction) with Existing Method.

Appname	Domain	Reduction technique	Total test cases	Execution time in seconds
Student Continuous Improvement	(0,30) (0,50) (0,40)	Test Case With No Reduction	64821	324105
		Test Case With Criteria Reduction	24149	12074.5
		Multi Constraint Based Reduction	31	15.5
Banking Business	(0,31) (32,55) (0,35)	Test Case With No Reduction	64512	32256
		Test Case With Criteria Reduction	27648	13824
		Multi Constraint Based Reduction	32	16
Business Promotion	(0,25) (0,60) (0,35)	Test Case With No Reduction	57096	28548
		Test Case With Criteria Reduction	32760	16380
		Multi Constraint Based Reduction	26	13

Table 9: Student Continuous Assessment.

Reg. no.	Name	Internal 1	Internal 2	Internal 3	Result	Total
124101	Raj	20	30	25	50 Marks Added	125
124102	Ravi	15	20	40	75 Marks Added	150
124103	Ram	35	25	10	25 Marks Added	95

Table 10: Banking Business.

Acc no	Custname	Bank Deposit 1	Bank Deposit 2	Bank Deposit 3	Result	Credits
134101501	Raj	18	15	35	50 credit points Added	50
134101502	Ravi	19	35	16	75 credit points Added	75
134101503	Latha	23	18	15	25 credit points Added	25

Table 11: Business Promotion.

Custname	Mobno	Pur1 Amt	Pur2 Amt	Pur3 Amt	Result	Credits
rana	9188476376	14	20	40	3rd purchase is greatest, so 25 marks added	75
Raju	8618109452	19	35	16	2nd purchase is greatest, so 50 marks added	50
ram	9946027073	23	18	15	First purchase is greatest, so 25 marks added	25

Apfd=1-(tf1+tf2+…tfm)/m*n+1/2*n for no order apfd=1-(4+4+1+2+2+3)/6*4+1/2*4=1-.666+.125=46%,for reverse order apfd=1-(1+1+4+3+3+2)/6*4+1/2*4=1-.5833+.125=54%

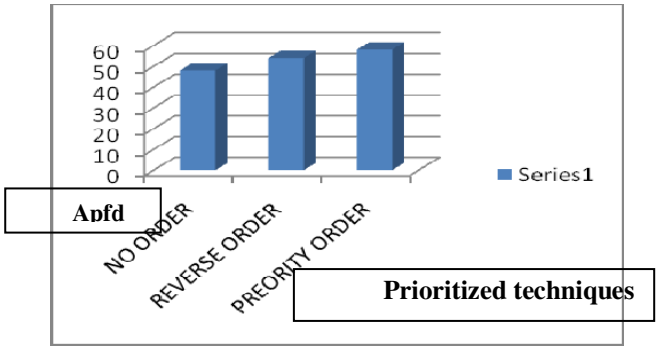

Fig. 2. Comparison between Different Prioritized Techniques no, Reverse and Proposed Order.

For proposed order we first calculate apfd=1-(1+1+1+1+4+4)/ 6*4+1/2*4=1-.4583+.125=58%,then we parallelize generated test cases by analyzing Table 2, now variable i used in 3 paths so range =total number of interval/3, and variable j used in 4 paths so range=total number of interval/4.

Since k is constant we not divide k, so range of I spitted into 3 parts i1) 0…..10 i2)11…..20 i3)21….30 . Similarly

range of j spitted into 4 parts j1) 10…..20 j2)21…..30 j3)31….40 j4)41…50. Now when we execute them in parallel fashion we have total number of test cases=[31*51*41]*4=259284 and reduced test case for path1=31, but using existing technique test cases=651, Reduced overall test cases= 31*1*1+10*41*1+21*31*1+1*51*1]=[31+410+651+51]=1 14, as in Table 2, then we assigning each test case constant .5 second and we observe that without parallelization execution time required for reduced test cases is 1143*.5=571.5.

So total number of test cases without test case reduction = 259284 and execution time=129642, but for sequential execution reduced test cases =1143, and execution time=571.5 and in case of PRIORITIZED parallel execution test cases=1143 and execution time=142.875, where processor having equal capacity and fault detection rate of proposed method is also more, and time required to detect all required faults will be considerably low because test cases are exposed in prioritized order as mentioned in Table 6.

VI. CONCLUSION

Every algorithm has its own dominance as well as hazard, ddr works on discrete domain and split points, ping pong and other prevailing approach results in immense number of test cases, compilation, time, vague effort and cost. But proposed technique has better performance by shrinking number of test cases, prioritizing them and betray vast number of bugs by designating test case rankings, apfd calculation and then deriving prioritized test suits, lessening test cases as in Fig. 2, and our proposed techniques is compared with many other existing techniques by application oriented research based comparison along with the evaluated result as in Table 8, ultimately we allocated the test case rankings to accomplish optimized performance and ultimately we execute parallelized and prioritized test cases to diminish overall running time, total budget required to perform testing.

VIII. LIMITATION

Methodology proposed by us inquire each and every path to prosecute serially to observe the control flow and each and every path is to be investigated to find out all possible curb, and need to examine all the possible criteria's related to the variables and analyze association between variables, so it will take supplementary time and memory to obtain the result, it is effective when the variables are there with constant and predetermined values and it works well for parallel execution where we have preserved memory and enlarge comprehensive speed of execution.

FUTURE SCOPE

To build a methodology which will consider requirements of user, and reduces number of test cases based on requirements of the user along with considering priority of the user requirements.

ACKNOWLEDGEMENT

First of all, i am grateful to the god for the good health and wellbeing that were necessary to complete this work. Next i would like to thank to my parents, guide, wife and all who helped me directly and indirectly to complete my research work.

Conflict of Interest: Nil

REFERENCES

[1]. Wang, R., Qu, B., & Lu, Y. (2015). Empirical study of the effects of different profiles on regression test case reduction. *IET Software*, **9**(2), 29-38.

[2]. Boehm, B., & Huang, L.G. (2003). Value-based software engineering: A case study. *Computer*, **36**(3), 33-41.

[3]. Arnicane, V. (2009). Complexity of equivalence class and boundary value testing methods. *International Journal of Computer Science and Information Technology*, **751**(3), 80-101.

[4]. Sawant, A.A., Bari, P.H., & Chawan, P.M. (2012). Software testing techniques and strategies. *International Journal of Engineering Research and Applications (IJERA)*, **2**(3), 980-986.

[5]. Jeng, B. and Weyuker, E.J. (1994). A simplified domain-testing strategy. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, **3**(3), 254-270.

[6]. Gyimóthy, T., Beszédes, Á., & Forgács, I. (1999). An efficient relevant slicing method for debugging. In *Software Engineering—ESEC/FSE'99* (pp. 303-321). Springer, Berlin, Heidelberg.

[7]. Biswas, S., Mall, R., Satpathy, M., & Sukumaran, S. (2011). Regression test selection techniques: A survey. *Informatica*, **35**(3).

[8]. Saifan, A.A., Alsukhni, E., Alawneh, H., & Sbaih, A.A. (2016). Test case reduction using data mining technique. *International Journal of Software Innovation (IJSI)*, **4**(4), 56-70.

[9]. Assi, R.A., Masri, W., & Zaraket, F. (2016). UCov: a user-defined coverage criterion for test case intent verification. *Software Testing, Verification and Reliability*, **26**(6), 460-491.

[10]. Kumar, A. (2016). Evaluation of software testing techniques through software testability index. *AKGEC,* Vol. **3**, pp. 342-349.

[11]. Srikanth, H., Williams, L., & Osborne, J. (2005). System test case prioritization of new and regression test cases. In *2005 International Symposium on Empirical Software Engineering, 2005.* 64-73.

[12]. DeMilli, R.A., & Offutt, A.J. (1991). Constraint-based automatic test data generation. *IEEE Transactions on Software Engineering*, **17**(9), 900-910.

[13]. Offutt, A.J., Jin, Z., & Pan, J. (1999). The dynamic domain reduction procedure for test data generation. *Software: Practice and Experience*, **29**(2), 167-193.

How to cite this article: Udupa, P. and Begum, A.R. (2019). An Integrated Methodology for Testing Source Code by Using Multi Constraint Reduction, Test Suite Prioritization and Prioritized Parallelization. *International Journal of Emerging Technologies*, **10**(2): 221–225.