Volume 5, Issue 4, 2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

A Unit – Test Case Prioritization Technique Based on Source Code Analysis

Harish Kumar*, Vedpal, Naresh Chauhan
Department of Computer Engineering
YMCA University of Science & Tech., India

Abstract— While performing the white box testing there may be large number of test cases executed by the developer to ensure the correct functionality of their code. This process involves a lot of efforts. To reduce these efforts if somehow a developer is able to get the prioritized order of the test cases which he/she is going to execute to ensure the correct functionality during the process of white box testing is a challenging task. Keeping this idea in mind a test case prioritization technique for unit testing is proposed in this research paper. The proposed technique is based on the analysis of the source code written by the developers. To show the effectiveness of the proposed approach it is compared with non prioritized and random approach. The APFD value obtained by proposed approach is more, showing the efficacy of the proposed approach.

Keywords— Test case prioritization, TCP, software testing, Basic path testing, Source code analysis

I. INTRODUCTION

Software testing is an important activity of the software development process. It involves lot of time and resources. Testing must be done by keeping in mind all the factors like quality, reliability, integrity, efficiency. Designing of suitable and efficient test cases is a challenging task. There are two main techniques of testing [2, 4] i.e. white box testing and black box testing. White-box testing or structural testing [2, 4] is typically focused on the internal structure of the program. In white box testing, structure means the logic of the program which has been implemented in the language code. Basis path testing is an important part of white box testing. It monitors the whole control structure of the program. Based on the control structure a flow graph is prepared and all the possible paths are covered and executed during testing. It is considered as a general criterion for detecting more errors as all statements and all branches are covered while testing.

While performing white box testing if a developer gets a prioritized test suite in advance to test his program, there may be more chances of detecting bugs early thereby reducing the cost and effort of the testing process. Keeping this view in mind, a test case prioritization approach for unit testing based on source code analysis is proposed in this paper.

Section 2 of this paper discusses the related work done in the area of test case prioritization, section 3 covers the proposed approach for test case prioritization, section 4 discusses the evaluation and analysis of the proposed approach and section 5 briefs the conclusion.

A. Basis Path Testing

It is one of the oldest structural testing techniques [3].This technique is based on the control structure of the program. On the basis of that control structure, a flow graph is developed and it is assumed that all possible paths can be covered at least once during testing. Here, modified version of path coverage criterion is used which is the most general criterion when compared to other logic coverage criteria. The problem with the path coverage is that program that contains loops can have an infinite number of possible paths and it's impractical to test all those paths. Basis path testing is the testing technique of selecting the paths providing a basis set of execution paths through the program.

An execution path is a set of nodes and directed edges in a flow graph that connects (in a directed fashion) the start node to a terminal node. Two execution paths are said to be independent if they do not include the same set of nodes and edges.

B. Cyclomatic Complexity

Cyclomatic complexity [1, 2] is software metric that provides a quantitative measure of the logical complexity of a program. When used in the context of a basis path testing method, the value computed for cyclomatic complexity defines the number of independent paths in a basis set of a program and provides with a upper bound for the number of tests that must be conducted to ensure that all the statements have been executed at least once**.**

An independent path is any path through the program that introduces at least one new set of processing statements or a new condition .When stated in terms of a flow graph, an independent path must move along at least one edge that has not been traversed before the path is defined.

There can be different basis set for a procedural design, so a measure called cyclomatic complexity is used to define the paths which must be looked only. The value of this provides us the upper bound for the number of independent paths that comprise the basis set.

II. RELATED WORK

Srivastava et al. [5] proposed an approach for identification of effective paths in control flow graph for software under test and prioritized the most feasible path to be executed first using ant colony optimization algorithm.

Kumar et al. [8] discussed the basis path testing as an imperative testing method in white box testing. Basis path testing focused on internal logic therefore it generates a feasible set of independent path present in source code, which is known as basis path. Out of these paths some may be not feasible.

Zhonglin et al. [6] proposed an improved approach for basis path testing. The proposed technique combines the baseline method with dependence relation analysis. The method proposed by them generated a set of linearly independent paths, termed as basis paths.

Qingfeng et al. [7] elaborated the work proposed by Zhonglin for selection of infeasible paths. In his research paper he proposed a new approach for selection of independent paths. To show the effectiveness of his proposed approach he illustrated his work on a triangle program.

Himanshi et al. [9] proposed a technique to prioritize the paths using ant colony optimization. The proposed approach allows tester to find out the probability for each path and priority of the shortest path comes out to be maximum.

Ahmed S.Ghiduk [10] proposed an ant colony optimization based approach for generating set of optimal paths to cover all definition-use associations in the program under test. This approach uses the ant colony optimization to generate suite of test-data for satisfying the generated set of paths. He also introduced a case study to illustrate his approach.

Manika Tyagi & Sona Malhotra [11] proposed an approach to prioritize regression test cases based on three factors which are rate of fault detection, percentage of fault detected and risk detection ability. The proposed approach is compared with different prioritization techniques such as no prioritization, reverse prioritization and random prioritization using APFD (average percentage of fault detected) metric.

III. PROPOSED WORK

The proposed approach works at two levels. At the first level control flow graph (CFG) of the source code is created and independent paths are determined by analysing the CFG. After finding the all the independent path test cases are selected on the basis of the covering of the independent paths. At the second level determined test cases are prioritized. To prioritize the test cases the programme structure has been analyzed. Based on this analysis some factors related to source code can be identified. These factors when considered may become the basis for test case prioritization. Further, all the factors cannot be at same level. So each factor has been given a weight accordingly. The considered factors and their corresponding weights are shown in Table 1. The factors weight shows the criticality of the factor in term of the probability of errors introduced by the factors.

Table 1. Factors considered and their weight

S.No.	Factors	Factor weight
1	Line of Code	.05
2	Type Casting	.15
3	Predicate Statement	.175
4	File Access	.15
5	Dynamic memory Allocation	.225
6	Number of Input Variable	.1
7	Number of Output Variable	.05
8	Assignment Statement	.1

Test cases are prioritized on the basis of a test case prioritization value (TCPV). The TCPV of a test case is determined by using the formula1. Higher the TCPV of the test case higher is the priority of the test case for execution.

$$TCPV_i = \sum_{j=1}^{8} (vfactor_{ij} * wfactor_j) \quad \text{----------(1)}$$

Where the vfactor is value of the factor covered by the i^{th} test case , wfactor is the weight of the j^{th} factor

The algorithm of the proposed approach is as shown in algorithm 1

```
Begin
Source code (S)
1. Create the control follow graph (CFG) of S.
2. Identify all the independent path of S.
3. Determine the test case covering the independent paths and create the non prioritized list of test cases T.
4. Let T' be the prioritized order of test cases
5. While (T not empty)
Begin
6. Identify the factors covered by the test case corresponding to independent path of test case.
7. Calculate the TCPV of the test case by applying the formula 1
8. Order the test cases in the decreasing order on the basis of the TCPV and let it T'.
9. T' be the new prioritized order of test cases.
End
10. Execute the test cases in prioritize order
End
```

Fig. 1 Algorithm 1 for test case prioritization

IV. EVALUATION AND ANALYSIS

For analysis and experimental verification of the proposed approach it has been applied on a program implemented in C programming language. The considered program has 154 lines of code and performs various operations such as add a new record, display the records and update records. Before applying the proposed approach some errors have been intentionally introduced in the program. The findings of the proposed approach are as shown below.

A. Control flow Diagram

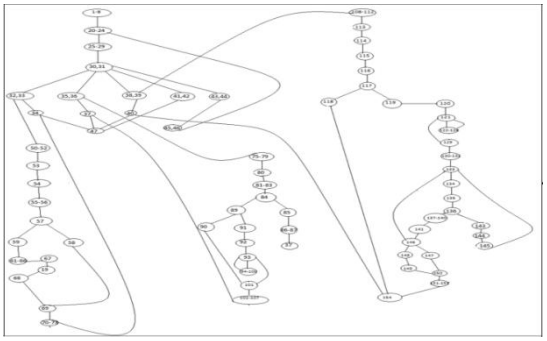

Fig.2 Control flow diagram of considered program

After analysis of the CFG all independent path are determined. Table 2 shows all independent paths of the sample program.

Table. 2 Independent paths of the considered program

S.No.	Path No.	Independent path
1	Path1	1-2-3-4-5-6-7-8-20-21-22-23-24-25-26-27-28-29-30-31-32-33-50-51-52-53-54-55-56-57-58-69-70-71-72-73-34-47
2	Path2	1-2-3-4-5-6-7-8-20-21-22-23-24-25-26-27-28-29-30-31-32-33-50-51-52-53-54-55-56-57-59-61-62-63-64-65-66-67-19-68-69-70-71-72-73-34-47
3	Path3	1-2-3-4-5-6-7-8-20-21-22-23-24-25-26-27-28-29-30-31-35-36-75-76-77-78-79-80-81-82-83-84-85-86-87-37-47
4	Path4	1-2-3-4-5-6-7-8-20-21-22-23-24-25-26-27-28-29-30-31-35-36-75-76-77-78-79-80-81-82-83-84-89-90-101-102-103-104-105-106-107-37-47
5	Path5	1-2-3-4-5-6-7-8-20-21-22-23-24-25-26-27-28-29-30-31-35-36-75-76-77-78-79-80-14-81-82-83-84-89-91-92-93-94-95-96-97-98-99-100-93-101-102-103-104-105-106-107-37-47
6	Path6	1-2-3-4-5-6-7-8-20-21-22-23-24-25-26-27-28-29-30-31-108-109-110-111-112-113-114-115-116-117-118-154-40-47
7	Path7	1-2-3-4-5-6-7-8-20-21-22-23-24-25-26-27-28-29-30-31-108-109-110-111-112-113-114-115-116-117-119-120-121-122-123-124-125-126-127-128-121-129-130-131-132-133-134-135-136-137-138-139-140-141-146-147-150-151-152-153-154-40-47
8	Path8	1-2-3-4-5-6-7-8-20-21-22-23-24-25-26-27-28-29-30-31-108-109-110-111-112-113-114-115-116-117-119-120-121-122-123-124-125-126-127-128-121-129-130-131-132-133-134-135-136-143-144-145-133-146-148-149-150-151-152-153-154-40-47
9	Path9	1-2-3-4-5-6-7-8-20-21-22-23-24-25-26-27-28-29-30-31-41-42-47
10	Path10	1-2-3-4-5-6-7-8-20-21-22-23-24-25-26-27-28-29-30-31-43-44-45-46-24-25-26-27-28-29-30-31-41-42-47

Table 3. shows the test cases which cover the all feasible independent paths.

Table 3. Test cases covered the independent paths

Test case id	Value1	Value 2	Result expected	Path followed
TC1	-	-	ERROR	P1
TC2	100	ABC	INSERTED	P2
TC3	-	-	NO RECORD TO DISPLAY	P3
TC4	-	--	ERROR	P4
TC5	-	-	100 ABC	P5

			200 XYZ	
TC6	-	-	ERROR	P6
TC7	100	MNW	UPDATE SUCCESSFUL	P7
TC8	250	-	ENTER CORRECT ID	P8
TC9	4	-	Exit	P9
TC10	5	-	ENTER CORRECT CHOICE	P10

After selecting the test cases corresponding to the all feasible independent path the factors are determined those covered by the test cases. Out of these test cases TC9 and TC10 are not consider because they do not cover any factors discussed in the proposed approach. The table 4 shows the factors covered by the testcases.

Table. 4 Factors covered by the test cases

S. No.	Factors	TC1	TC2	TC3	TC4	TC5	TC6	TC7	TC8
1	Line of code	10	17	14	12	21	11	39	32
2	Type Casting	2	2	2	2	2	2	2	2
3	Conditional statement	1	1	1	2	3	1	8	7
4	File Access	2	3	2	2	4	2	8	7
5	Dynamic memory allocation	2	2	2	2	2	2	2	2
6	No. of Input variable	0	2	0	0	0	0	2	2
7	NO. of Output variable	0	0	0	0	2	0	2	2
8	Assignment Statement	3	4	4	4	5	4	8	7

After determined the factors covered by the test cases TCPV for each case is calculated by using the formula 1. Table 5 shows the TCPV for each test case.

Table. 5 Calculated TCPV of Test cases

S. No.	Factors	TC1	TC2	TC3	TC4	TC5	TC6	TC7	TC8
1	**Line of Code**	5.	8.5	7.0	6.0	10.5	5.5	19.5	16
2	**Type Casting**	.3	.3	.3	.3	.3	.3	.3	.3
3	**Conditional Statement**	.175	.175	.175	.35	.525	.175	.875	.875
4	**File Access**	.3	.45	.3	.3	.6	.3	1.20	1.05
5	**Dynamic memory Allocation**	.45	.45	.45	.45	.45	.45	.45	.45
6	**Number of Input Variable**	0	.2	0	0	0	0	.2	.2
7	**Number of Output Variable**	0	0	0	0	.1	0	.1	.1
8	**Assignment Statement**	.3	.4	.4	.4	.5	.4	.8	.7
	TCPV	6.525	10.475	8.625	7.8	12.975	7.125	23.425	19.625

The prioritized orders of the test cases are TC7, TC8, TC5, TC2, TC3, TC4, TC6, TC1
The table 6 shows the fault detected by the test cases when test cases are executed in non prioritized order.

Table 6 .Faults Detected in Non prioritized order

	TC1	TC2	TC3	TC4	TC5	TC6	TC7	TC8
F1	*	*						
F2		*						
F3		*						
F4			*		*			
F5					*			

F6						*	*	*
F7							*	*
F8							*	*
F9							*	
F10						*	*	*

The table 7. shows the faults detected by the test cases when the test cases are executed in the prioritized order obtain by applying the presented approach

Table 7. Faults Detected in Prioritized order of test cases

	TC7	TC8	TC5	TC2	TC3	TC4	TC6	TC1
F1				*				*
F2				*				
F3				*				
F4			*		*			
F5			*					
F6	*	*					*	
F7	*	*						
F8	*	*						
F9	*							
F10	*	*					*	

The table 8 shows the faults detected by the test cases when they executed in the random order

Table. 8 Faults Detected in Random order of test cases

	TC3	**TC5**	**TC8**	**TC4**	**TC1**	**TC7**	**TC2**	**TC6**
F1					*		*	
F2							*	
F3							*	
F4	*	*						
F5		*						
F6			*			*		*
F7			*			*		
F8			*			*		
F9						*		
F10			*			*		*

By using the formula 2 the APFD for all approaches of test case prioritization were calculated as given below.
APFD = 1 – (TF1 + TF2 +TF3 +TF4+.....+ TFm) /(n*m) + 1/(2*n) ----------(2)
Where n is the number of faults and m is the number of test cases

A. APFD for the non Prioritized approach

APFD = 1- (1+2+2+3+5+6+7+7+7+6) / 80 + 1/(2*8)
= 1-(46/80) +1/(16)
= 48 %

B. APFD for the Random approach

APFD = 1- (5+7+7+1+2+3+3+3+6+3) / 80 + 1/(2*8)
= 1 – (40/80) +1/(16)
= 56 %

C. APFD for the proposed approach

APFD = 1- (4+4+4+3+3+1+1+1+1+1) / 80 + 1/(2*8)
= 1-(23/80) +1/(16)
= 78 %

The table 9 shows the APFD value when the test cases are executed in non prioritize order, in random order and in prioritized order obtain by applying the proposed approach.

Table 9. APFD Metric

S.No	Applied Technique	APFD
1	Non Prioritized	49%
2	Random approach	56%
3	Proposed approach	78%

The comparison of APFD graph of proposed approach, random approach, non prioritized approach as shown in graph1 shows the effectiveness of the proposed approach

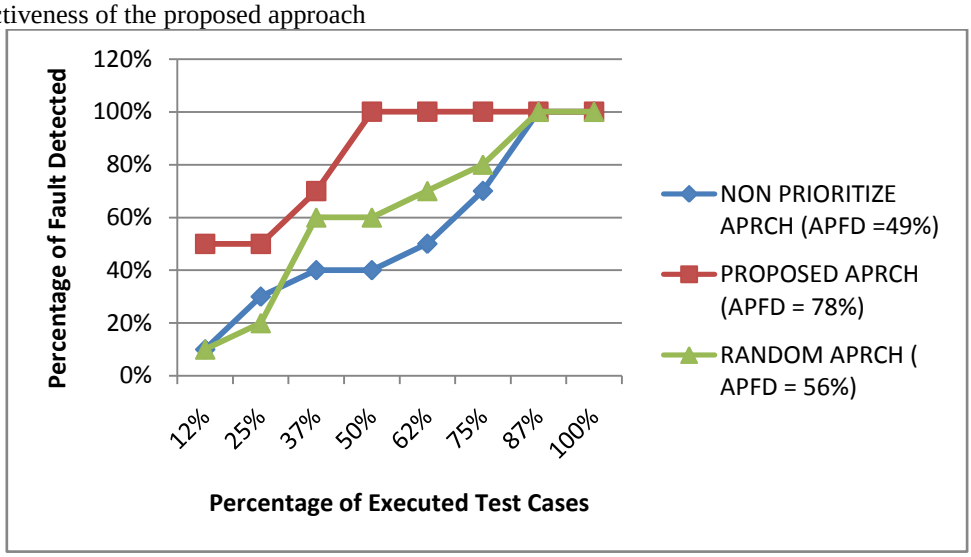

Fig. 3 Comparison of Proposed, Random and Non Prioritize approach

V. CONCLUSIONS

In this paper a test case prioritization technique has been proposed to prioritize the test cases of unit level testing. The presented approach works at two levels. At the first level independent paths and test cases covers the identified paths are determined. At the second level test cases are prioritized. For experimental validation and analysis of the proposed approach it has been applied on a program implemented in C programming language. The obtained result after applying the proposed approach shows the effectiveness of the approach.

REFERENCES

[1] G.J. Myers,The Art of Software Testing, John Wiley & Sons,1979.
[2] Dr. Naresh Chauhan ,"Software Testing – Principle and Practice" ,Oxford university press, 2010.
[3] T. McCabe, "A Complexity Measure", IEEE Trans. On Software Engg., 1976, pp308-320.
[4] Pankaj Jalote, "An Integrated approach to software engineering", Narosa Publishing House, Second Edition, 2003.
[5] Saurabh Srivastava,Sarvesh Kumar,Ajeet Kr. Verma, " Optimal path sequence in basis path testing", International Journal of Advanced Computational Engineering and Networking,ISSN(p):2320-2106,Volume-1,Issue-1,Mar-2013.
[6] Zhang Zhonglin, M.L., "An improved Method of acquiring basis path for software testing",ICCSE'10(pp.1891-1894).IEEE.
[7] Qingfeng, D. "An improved algorithm for basis path testing",2011 ,IEEE(pp. 175-178).
[8] T. Bharat Kumar, N.H., " A catholic and enhanced study on basis path testing to avoid infeasible paths in CFG".Global trends in information systems and software applications,2012.(pp. 386-395).
[9] Himanshi,Nitin Umesh and Saurabh Srivastva, " Path Prioritization using Meta-Heuristic Approach", International Journal of Computer Applications(0975-8887),Volume 77-No.11,September 2013.
[10] Ahmed S. Ghiduk, "A new software data-flow testing approach via ant colony algorithms", Universal Journal of Computer science and engineering technology, 1(1),64-72,oct 2010. ISSN: 2219-2158.
[11] Manika Tyagi & Sona Malhotra, "An Approach for Test Case Prioritization Based on Three Factors", I.J. Information Technology and Computer Science, 2015, 04, 79-86 Published Online March 2015 in MECS.
[12] Harish kumar and Naresh Chauhan " Identifying and analyzing the research challenges in the test case \ prioritization" International Journal of computer science and engineering system, pages 88-98 Volume No. 6(2012) Issue No. 3(2012)
[13] Harish Kumar , Vedpal and Naresh Chuhan , " A hierarchical system test case prioritization based on requirements" 13 th Annual International software testing conference in India, 2013 , 04 -05 December 2013, Banglore, india
[14] Harish Kumar and Naresh Chauhan " A Coupling effect based test case prioritization technique" 9th Indiacom 2nd International conference on computing for sustainable global development , 2015

