ISSN 22773061

Test Case Selection and Prioritization for Regression Testing using Fault Severity

Uma Sharma, Vedant Rastogi
M.Tech , IET, Alwar , Rajasthan, India
mailuma2006@gmail.com
Assistant Proffessor, IET , Alwar ,Rajasthan,India

ABSTRACT

Regression testing is a significant but a very expensive testing process .Test case prioritization is a technique to schedule and execute the test cases in such an order that results in increasing their ability to meet some performance goal. One of the main goal is to increase the rate of fault detection –i.e. to detect the faults as early as possible during the testing process. Test case prioritization is used to minimize the expenses of regression testing. This paper proposes a technique to select and prioritize the test cases and results in improving the rate of fault detection.

Indexing terms/Keywords

Regression Testing , Test case prioritization, Prioritization techniques..

Academic Discipline And Sub-Disciplines

Software engineering , Software Testing

SUBJECT CLASSIFICATION

Test Case Prioritization for Regression Testing

TYPE (METHOD/APPROACH)

Test Case Selection , Test case minimization ,Hybrid approach

Council for Innovative Research
Peer Review Research Publishing System
Journal: INTERNATIONAL JOURNAL OF COMPUTERS & TECHNOLOGY
Vol 9, No 3
editor@cirworld.com
www.cirworld.com, member.cirworld.com

INTRODUCTION

Software testing is a process to ensure that software is working properly or as required and is not doing anything unintended. Software testing is an important and costly phase of SDLC. It consumes over 50% cost of the entire software. Regression testing is performed to validate the modified software to ensure that the changes are correct and do not introduce additional errors . Regression test suite is typically large and it is very expensive to execute all the test suits during regression testing. That's why, test cases are prioritized for execution in order to increase their ability to meet some performance goals, e.g., to increase the rate of fault detection or to achieve maximum code coverage etc. Gregg Rothermal [1] has proven that prioritizing and scheduling text cases are one of the most critical task during the software testing process. He stated that the test case prioritization methods and process are required because (a) the regression testing phase consumes a lot of time and cost to run , (b) there is not enough time and resources to run the entire test suite, (c) there is a need to decide which test case to run first. The next section of this article describe the Problem Statement, then section 3 mentioned different Methodologies related to regression testing , next section 4 describe the Related Work and section 5 have the Proposed Work and at last we concluded with the summary.

PROBLEM STATEMENT

Rothermel at el. [1, 7] defines the test case prioritization problem as follows :
Given: T, a test suite; PT, the set of permutations of T; f, a function from PT to the real numbers.
Problem: Find T' belongs to PT such that (for all T") (T" belongs to PT) (T" ≠ T') [f (T') ≥ f(T")].
Here, PT represents the set of all possible prioritizations (orderings) of T and f is a function that, applied to any such ordering, yields an award value for that ordering [1,2].

METHODOLOGIES

This section provides the methodologies that are related to regression testing. There are four methodologies that are available for regression.testing. These methods are [2,5]:

- Retest all
- Regression Test Selection
- Test Suite Reduction
- Test Case Prioritization

Retest all

In this technique the test cases that are no longer apply to modified version of program are discarded and all the remaining set of test cases are used to test the modified program. Retest all technique takes time and effort as all test cases are used to test the program again, so may be quite expensive.

Regression Test Selection

This technique is much better than Retest All as it executes a subset of the test suite.It uses information about program, modified program, test cases to select a subset of test cases for testing.

Cost Effective-based techniques

This technique uses information about program and test suite to remove the test cases, which have become redundant with time, as new functionality is added. It is different from Regression test selection as former does not permanently remove test cases but selects those that are required. Advantage of this technique is that it reduces cost of validating, executing, managing test suites over future releases of software, but the downside of this is that it might reduce the fault detection capability with the reduction of test suite size.

Test Case Prioritization

In this technique each test cases are assigned a priority. Priority is set according to some criterion like faster code coverage or rate of faults detect etc, and test cases with highest priority are scheduled first .

RELATED WORK

Lots of research has been done and techniques are developed on test case prioritization for regression testing.
Empirical study on test case prioritization techniques reported in [1, 2] sorts test cases such that the test cases with highest priority, according to some criterion, are executed first. Test case prioritization can address a wide variety of objectives. For example, in [1] Rothermel prioritized the test cases based on coverage alone. Testers might wish to schedule, test cases in order to achieve code coverage at the fastest rate possible during initial phase of regression testing so as to reach 100% coverage at the earliest or to ensure that the maximum possible coverage is achieved by some pre–determined cut–off point.
In literature, many techniques for regression test case prioritization for general applications and test suite prioritization strategies for web applications have been described. Most of these techniques are code–based, relying on information, relating test cases to coverage of code elements. In [02], Elbaum investigated several prioritizing techniques such as total statement (or branch) coverage prioritization and additional statement (or branch) coverage prioritization that can improve the rate of fault detection. Dennis Jeffrey, Neelam Gupta [10] present an approach to prioritize test cases based on the coverage requirements present in the relevant slices of the outputs of test cases. S. Raju and, G. V. Uma propose a set of prioritization factors to design a requirement based system level test case prioritization scheme using Genetic Algorithm (GA). These factors may be concrete, such as test case length, code coverage, data flow, and fault proneness, or abstract, such as perceived code complexity and severity of faults, which prioritizes the system test cases based on the

six factors: customer priority, changes in requirement, implementation complexity, completeness, traceability and fault impact. In [09], Srivastava present a new test case prioritization algorithm, which calculates average faults found per minute. Non–coverage based techniques found in the literature include Fault–Exposing–Potential (FEP) prioritization [1], history–based test prioritization [08], and the incorporation of varying test costs and fault severities into test case prioritization [Elb01, Elb02]. Paolo have proposed a test case prioritization technique that takes advantage of user knowledge through a machine learning algorithm, Case-Based Ranking (CBR). CBR elicits just relative priority information from the user, in the form of pairwise test case comparisons.
In [04] Zhang Li, Mark Harman, and Robert M. Hierons studied five search techniques: two meta–heuristic search techniques (Hill Climbing and Genetic Algorithms), together with three greedy algorithms (Basic Greedy, Additional Greedy and 2–Optimal Greedy) and proved that Genetic.

PROPOSED WORK

This section provides the proposed algorithm. The algorithm is basically divided into two parts. In first part Regression test selection is performed than prioritization is done on the selected test cases using faults severity criteria in the second part.

Process Description

1) Consider every independent path of CFG as a test case.
2) Get no of faults in every test case from previous executions. Also get the severity defined for all the test cases .
3) Get execution time of all the test cases .
4) Store data (test cases id, faults id, fault severity execution time) in database.
5) Now, take test case id, fault id and execution time as input for the first step of Algorithm ie Test case Selection Process .
6) In Test Case Selection Process i.e. in First Step of Algorithm , the test cases which have redundant faults i.e. faults which are also covered in some other test cases are removed following some steps and rest are selected for execution..
7) Now, for second step take output of first step i.e. Selected Test Cases and previously defined severity of faults occurred in these test cases as input.
8) Now , suppose severity is assigned in terms of numbers 1 to 5 , 1 is for lowest severity and 5 is for high severity (severity is assigned on the basis of customer requirement).
9) Arrange the test cases in order having the test case which has highest number of high severity faults, first. That is, test case which has highest number of faults having severity 5. Assign highest priority to this test case.
10) Cover all the test cases in decreasing order of number of severity of faults from high to low severity.
11) In this way, the test cases will be arranged in decreasing order of priority given to them on the basis of number of fault severities.

The Algorithm

First step :
Input:

1) Test cases ID (TC1,TC2,TC3, TC4..)
2) Faults ID (F1,F2,F3,F4..)
3) Execution Time (t1, t2, t3, t4…)

Begin:

1) Select the test case which has highest no of faults [(TC max-fault)].
2) If there are more than one test case having same number of faults ,select the one which has less execution time .
3) Note the faults occurred in this test case .(TC (max faults)).
4) Match the faults of other test cases (TC(match)) , which have the same faults as in the (TC (max faults)).
5) If all the faults of any other test cases are covered in the TC (max faults) than we will not execute that test case(TC not selected) .
6) But if some faults are covered of any test case in the TC (max faults) and a few are left uncovered than remove the faults which are covered .
7) Now again from the list of test cases which have faults , excluding , previous TC (max faults) and TC (not selected), find the next TC (max fault).
8) Repeat from step 1 to step 7, till all the test cases are covered .This way , about 30% to 50% cases are covered in (TC not selected) means not to be executed .That is 50% to 80% test cases are required to prioritize .

Output:

Selected Test Cases: TC1, TC2...

Second Step:

Input:

1) Output of first step : Selected Test cases .
2) Faults Severity.

Begin:

ISSN 22773061

1) Now, from the SELECTED TEST CASES , Select the test case which has highest number of high severity faults (S5) as high priority test case.
2) This way , first cover all the test cases which have high severity faults (S5) ,in decreasing order of number of faults present of severity S5.
3) Next , select the test case which has highest number of faults having severity (S4).
4) This way , cover all test cases in decreasing order of severity of faults (ie from S5 to S1) following decreasing order of number of faults in the same.

OUTPUT : TC2,TC1……

RESULT ANALYSIS

To validate the proposed Algorithm, the metric APFD (Average Percentage of Faults Detected) is used. [6], It measures the weighted average of the percentage of faults detected over the life of the suite. APFD values range from 0 to 100; higher number simply faster (better) fault detection rates.
Let T be a test suite containing n test cases and let F be a set of m faults revealed by T.
Let TFi be the first test case that reveals fault i for a given order of the test cases in the test suite T.
The APFD for test suite T is calculated using equation

$$\text{APFD} = 1 - \frac{TF_1 + TF_2 + \ldots + TF_m}{nm} + \frac{1}{2n}$$

EXAMPLE :

Input (from previous executions)

Table 1. Input Table for Step 1 of Proposed Algorithm [9]

Test Case/ Faults	TC1	TC2	TC3	TC4	TC5	TC6	TC7	TC8	TC9	TC10	SEVERITY
F1								*	*		4
F2		*	*		*						2
F3				*		*				*	2
F4		*	*								1
F5								*			3
F6								*	*		2
F7				*	*		*				2
F8	*					*					2
F9				*		*				*	1
F10	*							*			1
No of Faults	2	2	2	3	2	3	1	4	2	2	
EXE TIME	9	8	14	9	12	14	11	10	10	13	

Example computation :

1) Select test case having max. no of faults :
   tc8 is selected : f1, f5, f6, f10
2) Match the faults of selected test case with other test cases
   tc9 : f1, f6 (all faults covered) Rejected test case
   tc1 : f8 , f10 (fault f10 removed)
3) Next test case having max no of Faults : tc4, tc6

   tc4 having min execution time is selected

   tc4 : f3, f7, f9
   tc7 : f7 (all faults covered) Rejected Test Case
   tc9 : f3 , f9 (all faults covered) Rejected Test case
   tc6 : f3, f8 , f9 (faults f3, f9 removed)
   tc5 : f2 , f7 (fault f7 is removed)

4) Next test case having max no of faults : tc1 , tc2, tc3

   tc2 having min. exection time is selected

   tc2 : f2 , f4
   tc3 : f2 , f4 (all faults covered) Rejected Test case
   tc5 : f2 (all faults covered) Rejected Test case

5) Next test case having max. No of faults : tc1 , tc6
   tc1 selected having min. execution time
   tc1 : f8
   tc6 : f8 (all faults covered) Rejected Test case

Step 1 : Output :

Selected Test Case : tc1 , tc2, tc4, tc8
Rejected test case : tc3, tc5, tc6, tc7, tc9

Step 2:

Input :

Selected Test Case : tc1 , tc2, tc4, tc8
Select test case having maximum no of highest severity faults.

Table 2: Input for second step of algorithm

Test Case/ Faults	TC1	TC2	TC4	TC8	SEVERITY
F1				*	4
F2		*			2
F3			*		2
F4		*			1
F5				*	3
F6				*	2
F7			*		2
F8	*				2
F9			*		1
F10	*			*	1
No of Faults	2	2	3	4	

1) tc8 has fault having max. severity
   Prioritized test case : tc8
2) Next test case selected is tc4 (having 2 faults of highest severity (2)).
   Prioritized test case : tc4
3) Next test case selected is tc1 :
   Prioritized test case : tc1
4) Next test case selected is tc2 :
   Prioritized test case : tc2

Final output : tc8 , tc4, tc1, tc2

$$APFD = 1 - \frac{1+4+2+4+1+1+2+3+2+3}{10*10} + \frac{1}{2*10}$$

$$= 1 - 23\backslash 100 + 1\backslash 20$$

$$= .72$$

Comparison between the technique implemented and other related work is shown in Table 3

Table 3: Comparison Between the Proposed Technique and Other Related Work

Test case Order		APFD
Non Prioritized Order	**tc1, tc2 ,tc3, tc4 ,tc5, tc6, tc7, tc8, tc9, tc10**	**60**
Random Order	**tc7, tc10, tc5, tc2, tc3, tc1, tc6, tc4, tc9, tc8**	**47**
TCP for RT based on severity of Fault	**tc8, tc4, tc9, tc6, tc5, tc2, tc1, tc10, tc3, tc7.**	**70**
Prioritized order by average faults found per minute algorithm	**Tc8,tc4,tc2,tc1,tc6,tc9,tc5,tc10,tc3,tc7**	**72**
Proposed Algorithm	**tc8, tc4,tc1,tc2**	**72**

ACKNOWLEDGMENTS

We would like to express our heartiest gratitude to all the people who poured their efforts in compilation of this work. We would like to thank almighty for giving us strength to pull through this task and to all the individuals who gave their best contribution in the related field of research.Our thanks to the experts who have contributed towards development of the template.

REFERENCES

[1] Gregg Rothermel, Roland H. Untch, Chengyun Chu and Mary Jean Harrold, "Prioritizing Test Cases for Regression Testing", IEEE Transactions on Software Engineering, 2001.

[2] S. Elbaum, A. Malishevsky, and G.Rothermel Test case prioritization: A family of empirical studies. IEEE Transactions on Software Engineering, February 2002.

[3] Hema Srikanth and Laurie Williams,"Requirements-Based Test Case Prioritization",North Carolina State University, ACM SIGSOFT Software Engineering, pages 1-3, 2005.

[4] Xiaofang Zhang, Baowen Xu, Changhai Nie and Liang Shi, "An Approach for Optimizing Test Suite Based on Testing Requirement Reduction", Journal of Software (in Chinese), 18(4): 821-831, 2007.

[5] David Leon and Andy Podgurski, "A Comparison of Coverage-Based and Distribution-Based Techniques for Filtering and Prioritizing Test Cases", *Proc. Int'l Symp. Software Reliability Eng.*, pp. 442-453, 2003.

[6] Alexey G. Malishevsky, Joseph R. Ruthruff, Gregg Rothermeland Sebastian Elbaum, "Cost-cognizantTest Case Prioritization", Technical Report TRUNL-CSE-2006-0004, Department of Computer Science and Engineering, University of Nebraska –Lincoln, 2006.

[7] H. K. N. Leung and L. White, "A cost model to compare regression test strategies", In *Proc. Conf. Softw. Maint.*, pages 201–208, 1991.

[8] Jung-Min Kim and Adam Porter, "A History-Based Test Prioritization Technique for Regression Testing in Resource Constrained Environments", In*Proceedings of theInternational Conference onSoftware Engineering (ICSE)*,pages 119–129. ACM Press, 2002.

[9] Praveen Ranjan Srivastava "Test Case Prioritization" Computer Science and Information System Group, BITS Pilani, India, Journal of Theoretical and Applied Information Technology JATIT, 2008.

[10]Dennis Jeffrey, Neelam Gupta, "Test Case Prioritization Using Relevant Slices",2006.

Author' biography with Photo

Uma Sharma , MTech in Information Technology from IET, India. She has 3 years of teaching experience .Her field of interest are Software Engineering , computer networks, DBMS, Manets.

Vedant Rastogi is PHD and working as Assistant Professor with IET , India. His field of expertise are Compuer Networks, Software Engineering and Data Mining .