Volume 4, Issue 6, June 2014 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Test case Prioritization based on testing requirement Priorities and fault dependency

Himanshu Ghetia *
SCSE VIT University
India

Shantanu Santoki
SCSE VIT University
India

Vairamuthu S.
SCSE VIT University
India

Abstract— Test-case prioritization is effective approach in regression testing, test-case prioritization is nothing but scheduled the test-case in such a way so that we can get increase performance of testing and ability of testing to meet some goal, such as code coverage rate of fault detection. So many technique are available for test-case prioritization and all this techniques and metrics usually assumed that testing requirement priorities and test case costs are uniform in this paper, we present new test-case prioritization technique and associated metric based on verifying testing requirement priorities, we are trying to increase testing quality and customer satisfaction.

Keywords— Software Testing, Regression Testing, Test Case Prioritization, Priority.

I. INTRODUCTION

Regression testing is costly testing process used to validate new features and detect regression faults, which occurs continuously during the SDLC (software development lifecycle). Regression test suites are often simply test cases that software engineers have previously developed, and that have been saved so that they can be used later to perform regression testing[12]. Software engineering often save test suits because reuse these suits later in regression testing, because of limited resource and constrain it is not possible to execute all test cases in every testing iteration. Therefore tester must give some order for test case according to their priority like higher priority test-case executed earlier, priority assign according to some criterion.

Test case prioritization is nothing but schedule test-case in order to increase their effectiveness at meeting some performance goals, such goal are code coverage, rate of fault detection. During software testing independent fault can directly detected and removed but dependent fault cannot removed directly this type of fault removed only if leading fault is removed. There are so many prioritization technique are available numerous prioritization is one of the prioritization technique, in this first dimension along which these techniques can be distinguished is in terms of the type of code elements they consider. For example, statements, basic blocks, or functions. The second dimension involves the use of "feedback". For example, additional function coverage prioritization adjusts the coverage information for remaining test cases using the feedback information. The third dimension is the use of other sources of information. For example, test cost, fault severity or fault propagation probability[6].

For example one software takes limited number of input and function generate several type of output, single fault in input may cause large number of fault in output module if they are dependent. That's why in regression testing test-case which has fault in output module is executed first and then other it will delayed time and take long time to detect the original cause of faults, If the dependencies can be detected earlier in regression testing then debugging can be started earlier and fault removal time will improve.

To measure the performance of prioritized test suite, some metrics are provided, such as total statement coverage, additional function coverage, rate of fault detection and so on. Most of the prior techniques assume that testing requirement priorities and test-case cost are uniform however in real-time testing requirement and test-case cost is very widely. The testing requirement priority change frequently during SDLC process and another thing is that test cases usually require different execution time and resources.

So both the entity impact on the test-case prioritization.

II. PROBLEM STATEMENT

Elbaum et al formally defined the test case prioritization problem as follows [2]:
Given that: T, a test suite; PT, the set of permutations of T; f, a function from PT to the real numbers.
Problem: Find T$\in$ 'PT such that ($\forall$T") (T"$\in$ PT)
(T"$\neq$T')[f(T') $\geq$f(T")].
Here, PT is the set of all possible orders of T, and f is an objective function which used to any such order, gives an award value for that order [1]. Our goal to research this area to find the metric which gives the rate of dependency detection between the faults and provide the algorithm which priorities the test cases and gives the higher priority test case based on the test case requirements.

III. RELATED WORK

Many works are done in recent years to find the test case priorities based on some qualities or properties.[57]. Srivastava et al discovered a test case prioritization technique based on basic block coverage for large systems [10]. Wong et al proposed a hybrid technique that combines modification, minimization and prioritization-based selection using source code and test history [11]. Elbaum and Rothermel worked on different prioritization techniques. They taken base as a source code and test history from that they prioritization techniques can significantly improve rate of fault detection [5-7, 8-9]. Zhang et al prioritizes test cases based on varying testing requirement priorities and test case costs [2]. In [4],[5], a metric APFD is proposed for measuring rate of fault detection as a means of objective function and prioritization techniques such as total statement coverage and additional statement coverage are discussed to improve the rate of fault detection. This metric and these techniques, however, assume that all test case and fault costs are uniform [1]. Jeffrey and Gupta prioritize test cases using relevant slices [3].

IV. PROPOSED WORK

A. Prioritization technique and metric

In this section, two types of TCP_RP_TC are proposed, and then the metric MRP_TC is presented. Next, we discuss the estimation of requirement priority and test cost[13].Suppose the set of testing requirements R={r1,r2,…,rm}, test suite T={t1,t2,…, tn}, we assume that for every requirement r_R, there is at least one test case t in T that satisfies r. Testing requirements are determined by testing objective. Testing requirements can be different types of code elements, such as statements, basic blocks, functions; or features and attributions of system; or faults in system. Then, the test history includes:

(1) The satisfiability relation of T and R, denoted as Sat(T,R)n×m: if tj satisfies ri, then Sat(tj,ri)=1, otherwise Sat(tj, ri) = 0, here 1SiSm, 1SjSn.

(2) Test history of ri, denoted as riHistory(RH1, RH2,…,RHs), where RHi(1SiSs) is the term of RI History, such as requirement volatility, fault proneness of the requirement, customer-assigned priority.

(3) Test history of tj, denoted as tjHistory(TH1,TH2,…, THt), where THi(1SiSt) is the term of tjHistory, such as execution time, the number of faults detected.

Basing on Sat(T,R) n×m, we use Test(ri) to denote the set of all test cases that can satisfy requirement ri, where Test(ri)={t| Sat(t, ri) =1}. Similarly, Req(tj) is used to denote the set of all requirements that satisfied by test case tj, where Req(tj)={r| Sat(tj, r) =1}. Moreover, Test history of R, T can be calculated by riHistory, tjHistory respectively, where

$\bigcup_{i=1}^{m} riHistory\ , THistory = \bigcup_{j=1}^{n} tjHistory$

We add another attribute to existing algorithm TCP_RP_TC_Addtl is based on additional requirement coverage prioritization technique and

TCP_RP_TC_Total is total requirement prioritization. AddtlRPriority(tj) is the ratio of unsatisfied requirement priorities covered by tj to the total requirement priorities not yet covered, TCost(tj) is the test case cost of tj, then

the priority of test case tj, denoted as TPriority(tj), is:[1]

$$TPriority(t_j) = AddtlRPriority(t_j) / TCost(t_j)$$

The larger AddtlRPriority(tj) and the smaller TCost(tj) is, then the larger TPriority(tj) is.When the requirement priorities are not considered, AddtlRPriority(tj) is equal to the ratio of number of unsatisfied requirements covered by tj to the total number of requirements not yet covered, denoted as AddtlRCov(tj); When all the test costs are equal, TPriority(tj) is decided by

AddtlRPriority(tj); when both requirement priorities and test costs are identical, TPriority(tj) only depends on AddtlRCov(tj), in this situation, TCP_RP_TC_Addtl is equivalent to the conventional technique that based on additional requirement coverage. The relationship of these four situations mentioned above is displayed in Figure 1[1]**.**

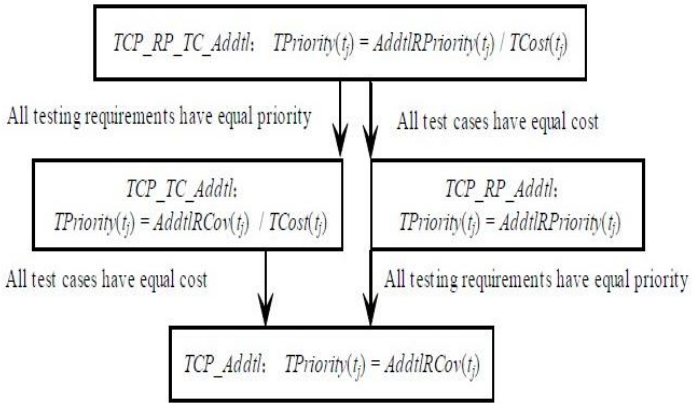

Figure 1The relationship of four situations in

TCP_RP_TC_Addtl [2]

We give priority to test-cases TCP_RP_TC_Addtl will order the test-case using Duple sort algorithm which describe below

Algorithm 1

Duple-Sort **Input:** T={t1, t2, …, tn},
TPriority[n]: set of priorities for each test case,
Req[n]: set of requirements that each test case satisfies
Output: SortedT: prioritized test suite
1. SortedT:=Sort (T, TPriority[n]); //descending order 2. if (exits test cases that have equal priority in SortedT) then
3. SortedT:=Sort (SortedT, |Req[n]|); //ascending order
4. endif

Duple algorithm first sort test-case(T) in decending order based on their priorities, if there exist some testcse which hase same priority than algorithm sort assiding order based on the Req(t) of each test-case, when priority are same, the test-case with smallest req(t),which cover highest requirement priorities and which was selected first, using this algorithm requirement whith higher prorities is executed as soon as possiable benifite of this approch is custormer satifaction ratio is good,time complexsity of duple algorithm is $O(n^2)$.

B. Measuring Effectiveness

To measure the efficiency of any software first as we described test case with its basic priority. From that priories test case suite we can quantify the how the test case will dependent to each other based on faults, for all this measurement objective function require. For measuring purpose we had taken a metric APFDD.

APFDD represents the "Average of the Percentage Fault Dependency Detected" for the implementation of the test suite. Its values range from 0 to 100 and higher value gives the faster fault dependency detection.

For the analysis we have taken one example. Consider the problem with the five faults and five test cases as a (F1, F2, F3, F4, and F5) and (T1, T2, T3, T4, and T5) respectively. We can represent the fault dependency as a directed graph G (V, E). Here the vertex V indicates the faults as V={V1, V2, V3, V4, V5). The edge set E is indicating the fault dependencies in which edge (F1, F2) represents the fault dependency between F1 and F2. Here (F1, F2) describes the F1 is dependent on F2. In our problem statement we consider six dependencies E= {(F2, F4), (F2, F5), (F4, F3), (F5, F1), (F5, F3), (F5,
F4))}. In figure 1 shows the dependency graph

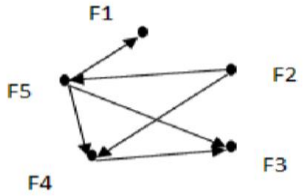

Figure 2 Dependency Graph [1].

We can also represent the give graph as the Dependency matrix given as the Table 1.

	F1	F2	F3	F4	F5
F1	0	0	0	0	0
F2	0	0	0	1	1
F3	0	0	0	0	0
F4	0	0	1	0	0
F5	1	0	1	1	0

Table 1 Dependency Graph [1].

Now we can calculate the NFD (F) = Number of faults dependent on fault F. So here NFD (F1) = 1, NFD (F2) = 0, NFD (F3) = 2, NFD (F4) = 2, NFD (F5) = 1.

Now consider the test cases with fault detecting abilities from Table 2.

	F1	F2	F3	F4	F5
T1	*				
T2		*			
T3		*			*
T4			*		
T5	*			*	*

Table 2 Test Case and Exposure of Fault [1].

As per the our test case priority we can described our test case as the ascending order consider as T1-T2-T3T4-T5 to form a prioritized test suite T'. "Fig. 2" shows the percentage of fault dependencies detected versus the fraction of T' used. After running T1 one dependency is detected as just one fault F1 is exposed and NFD (F1) =1, that mean after executing 20% of T' 16.67% dependencies are detected. Similar way after 40%, 60%, 80% and 100% execution of T' 16.67%, 50%, 83.33% and 100% fault dependencies are detected respectively.

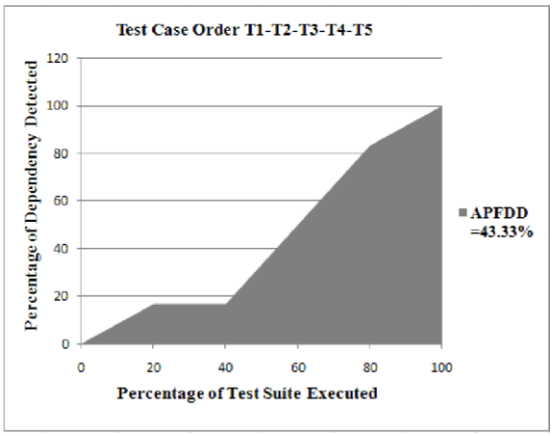

Figure 3 APFDD graph for non-prioritized test suite[1].

C. New Prioritization Technique

Prioritizing test case based on the fault dependency, it discovered maximum dependencies first. For improving the performance of the testing, software engineer should start their rectifying activates on those faults which growing later, speeding the release of the software. These faster detection methods provide the fast feedback on the software, and also provide beginning evidences when the quality goals have not beet satisfy. The techniques which present test suite prioritization algorithm that prioritizes the test cases with the goal of maximizing the number of faults dependency detection that are likely to be found during the execution of the prioritized test suite.

Algorithm 2:

Input: Test suite T, Fault dependency matrix M, fault F

Output: Prioritized test suite T'

1: begin 2: set T' empty
3: set NFD empty
4: set TDC empty 4: for each fault f'ε F do
5: for each fault f'' ε F do
6: if M [f', f''] =1 then 7: NFD [f''] =NFD [f''] +1
8: end if
9: end for
10: end for
11: for each test case t ε T do
12: for each fault f ε F do
13: if f first exposes in T then
14: TDC[t] =TDC[t] +NFD[f]
15: end if
16: end for
17: end for
18: sort T descending order based on the TDC value of each test case 19: let T' be T
20: end

The algorithm first determines the NFD of each faults based on the fault dependency matrix M. It then computes total dependency count (TDC) of each test case. TDC of a test case is simply the summation of NFD of faults that first expose in the test case. Using the value of TDC the algorithm then sorts the test cases in descending order of TDC value.

V. ANALYSIS

We analysis requirement priority distributions While TC=equal, RP= {equal, random}, result shown in Table 1. From the table we can find out that: the average M_{RP_TC} across different distributions TC_addti is larger than TC_ini is larger than TC_Total. As the impact of varying requirement priority distribution on the performance of prioritized test suites, Compared with B_1, the average .

MRP_TC of TC_Addtl in B2 is slightly higher, while that of Initial-Ordering is roughly equal, and that of TC_Total is slightly lower.

TC = equal	Addtl	Total	Ini	All(TCP)
B_1: equal	0.9481	0.9005	0.9164	0.9217
B_2: random	0.9510	0.8981	0.9163	0.9218
All(TC)	0.9496	0.8993	0.9164	

Table-3 Average MRP_TC for varying RP, TC = equal

As we mention above in our approach we can taken the example of the 5 faults and 5 test case generation in that we can if priorities the test cases as T5-T4-T2-T1-T3. This prioritized test suite has faster dependency detection rate with APFDD value of 83.33% as show in figure 3.

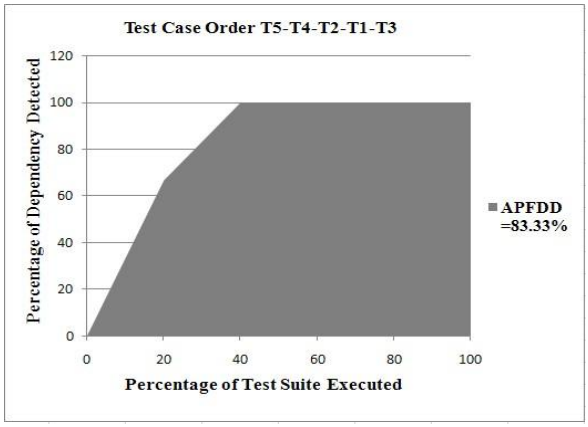

Figure 4 APFDD graph for prioritized test suite [1].

VI. CONCLUSION

Test case prioritization technique is widely used in regression testing to improve testing efficiency. In this technique trying to incorporate verifying testing requirement priorities to old technique in our technique TCP_RP_TC_Addtl and TCP_RP_TC_Total, adapted from previous work, and incorporate verifying testing requirement priorities incorporate and adjust with other prioritization techniques. The metric which evaluate "units-of-testing-requirement-priority-satisfied-per-unit test-case-cost" can provide the guideline for tester to select suitable test-case. This paper we can discus about first priorities the test case and then we measure effectiveness of test case prioritization in regression testing, based on that we discovered the dependency of faults. As above discus we mention the graph which described the test case is more effective. Here we can assume the earlier metric which is uniform but in real world it varies.

REFERENCES

[1] Md. Imrul Kayes,"Test Case Prioritization for Regression Testing Based on Fault Dependency", IEEE Conference on Quality Software,2011.

[2] X. Zhang, C. Nie, B. Xu, and B. Qu, "Test case prioritization based on varying testing requirement priorities and test case costs," Proc. International Conference on Quality Software, 2007, 15–24.

[3] D. Jeffrey and N. Gupta, "Test case prioritization using relevant slices," Proc. Computer Software and Applications Conference, 2006, 411–420..

[4] S. Elbaum, A. Malishevsky, and G. Rothermel, "Prioritizing test cases for regression testing," Proc. The 2000 ACM SIGSOFT International Symposium on Software Testing and Analysis, Portland, Oregon, U.S.A., August 2000, 102–112.

[5] S. Elbaum, A. Malishevsky, and G. Rothermel,"Test case prioritization: A family of empirical studies," IEEE Transactions on Software Engineering, vol. 28(2), 2002, pp. 159–182.

[6] S. Elbaum, G. Rothermel, S. Kanduri and A. G. Malishevsky. Selecting a Cost-Effective Test Case Prioritization Technique. Software Quality Journal, 12(3):185-210,2004.

[7] M. J. Harrold. Testing: A Roadmap, In Proceedings of the International Conference on Software Engineering, Limerick, Ireland, pages 61-72,2000.

[8] G. Rothermel, R. H. Untch, C. Chu and M. J. Harrold. Test Case Prioritization: An Empirical Study. In Proceedings of the International Conference on Software Maintenance, Sep 1999.

[9] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold. Prioritizing test cases for regression testing. IEEE Transactions of Software Engineering, 27(10): 929–948,2001.

[10] A. Srivastava and J. Thiagarajan. Effectively prioritizing tests in development environment. In Proceedings of the International Symposium on Software Testing and Analysis, pages 97-106, July 2002.

[11] W. E. Wong, J. R. Horgan, S. London and H. Agrawal. A Study of Effective Regression Testing in Practice. In Proceedings of the 8th IEEE International Symposium on Software Reliability Engineering, pages 264-274,Nov1997.

[12] Alexey G. Malishevsky, Joseph R. Ruthruff, Gregg Rothermel, Sebastian Elbaum, "Cost-cognizant Test Case Prioritization," Technical Report TR-UNL-CSE-2006-004, Department of Computer Science and Engineering, University of Nebraska–Lincoln, Lincoln, Nebraska, U.S.A., March 2006.

[13] B. Boehm. Value-Based Software Engineering. ACM SIGSOFT Software Engineering Notes, 18(2):3-7,2003

