Review of Prioritization Testing in Software Testing

Jyoti[1], Kirti Bhatia[2]
[1]*Mtech Scholar, Sat Kabir Institute of technology & Management, Bahadurgarh*
[2]*Assistant Professor, Sat Kabir Institute of technology & Management, Bahadurgarh*

Abstract—Requirement prioritization techniques have more pragmatic reference then the theoretical aspects and it differs with the domain and the organizations. Different BAs consider different aspects and follow various approaches in the process of prioritizing the requirements on the premise of different aspects of prioritization.

Prioritization is the essential skill that you need to make the very best use of your own efforts and those of your team. It's also a skill that you need to create calmness and space in your life so that you can focus your energy and attention on the things that really matter. Prioritization based on project value or profitability is probably the most commonly-used and rational basis for prioritization. Whether this is based on a subjective guess at value or a sophisticated financial evaluation, it often gives the most efficient results. Time constraints are important where other people are depending on you to complete a task, and particularly where this task is on the critical path of an important project. Here, a small amount of your own effort can go a very long way.

Keywords—prioritization technique, regression testing, efficiency

I. INTRODUCTION

Test case prioritization techniques organize the test cases in a test suite, prioritize them increase in the effectiveness of testing on the basis of requirements. One performance goal, the severe fault detection rate, is a measure of how quickly severe faults are detected during the testing process. An improved rate of severe fault detection can provide faster feedback regarding the quality of the system under testing, as complete testing process is vast and too expensive. This is often the case with regression testing, the process of validating the modified version of software to detect whether new errors have been introduced into previously tested code and to provide confidence that modifications are correct.

By increasing the overall rate of severe fault detection, a greater number of errors can be found more rapidly in the code developed to meet user requirements. As frequent rebuilding and regression testing achieves popularity, the need for a time constraint aware prioritization technique developing as per requirements. New software development processes such as extreme programming also promote a short development and testing cycle and frequent execution of fast test cases. Therefore, there is a clear need for a prioritization technique that has the potential for more effectiveness when a test suite's allowed execution time is known, particularly when that execution time is short.

This literature review shows that if the maximum time allotted for execution of the test cases is known using historical data of testing, a more effective prioritization can be produced. The time constrained test case prioritization problem can be effective and reduced to the NP-complete zero/one knapsack problem. This can often be efficiently approximated with a genetic algorithm (GA) heuristic search technique. Genetic algorithms have been effectively used in other software engineering and programming language problems such as test generation, program transformation, and software maintenance resource allocation, this survey demonstrates that they also prove to be effective in creating time constrained test prioritizations using requirements factors and technique that prioritizes regression test suites so that the new ordering:-

1. Will always run within a given time limit and
2. Will have the highest possible potential for severe defect detection based on derived coverage information and requirements. In summary, the important contributions of this survey are as follows:

- (i). a GA based technique to prioritize a regression test suite that will be run within a time constrained execution environment.
- (ii). an empirical evaluation of the effectiveness of the resulting prioritizations in relation to (i) GA-produced prioritizations using different user requirement parameters.

II. LITERATURE STUDY

This section discusses an overview of a software development life cycle (or SDLC) and a general software testing process. It describes a comprehensive set of existing test case prioritization methods researched from 1998 to 2008. In addition, it introduces a new "4C" classification of existing test case prioritization techniques. In general, the SDLC process contains the following phases, which are: requirement gathering, design & analysis, development, testing and maintenance. Those phases can be represented as follows in fig.1

From the above, the testing phases contain the following processes: test planning, test development, test execution and evaluation of results.

With existing test case prioritization techniques researched in 1998-2015, this paper introduces and organizes a new "4C"

classification of those existing techniques, based on their prioritization algorithm's characteristics, as follows:

1. Customer Requirement-based techniques:-Customer requirement-based techniques are methods to prioritize test cases based on requirement documents. Many researchers have researched this area, such as Srikanth, Zhang and Nilawar. Also, many weight factors have been used in these techniques, including custom-priority, requirement complexity and requirement volatility.
2. Coverage-based techniques:-Coverage-based techniques are methods to prioritize test cases based on coverage criteria, such as requirement coverage, total requirement coverage, additional requirement coverage and statement coverage. Many researchers have researched this area, such as Leon, Rothermel, and Bryce.
3. Cost Effective-based techniques:-Cost effective-based techniques are methods to prioritize test cases based on costs, such as cost of analysis and cost of prioritization. Many researchers have researched this area, for instance, Malishevsky,Alexey, and Elbaum.
4. Chronographic history-based techniques:-Chronographic history-based techniques are methods to prioritize test cases based on test execution history. A few researchers have researched this area.

b) Kristen R. Walcott Mary Lou Soffa, Gregory M. Kapfhammer Robert S. Roos, "Time Aware Test Suite Prioritization," ISSTA'06, July 17–20, (2006), Portland, Maine, USA[18]:-

- In this fitness based on(using GA):
  - The line of code coverage.
  - the time at which each test covers its associated code.

**But does not based on requirements prioritization required by user.Sujata, Mohit Kumar, Dr. Varun Kumar ,"Requirements based Test Case Prioritization using Genetic Algorithm," IJCST Vol. 1, Issue 2, December 2010.[17]:-

- In this fitness based on(using GA):
  - Requirement factors but does not satisfy the 80/20 rule (80% of Defects are Caused by 20% of Code).

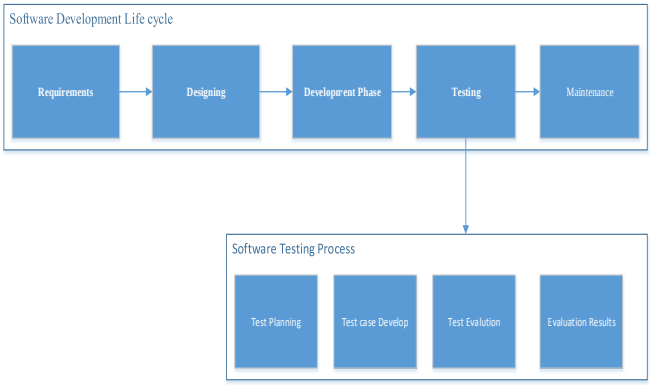

Fig.1:-Software Development Life Cycle

III. REQUIREMENT OF PRIORITIZATION

To find out the severity faults in less time, we can use the requirements and prioritize them in before developing cycle. Using this prioritization, we can easily fulfill the requirements of the user/customer in development phase.By knowing all issue regarding requirements, we can resolve interface requirement as user required so that less changed are required after implementing the code. Some techniques used for requirement prioritization as shows in fig. 2 below:

Test cases in the test suites are reschedule which will further prioritize using an algorithm. Before dealing with prioritization algorithms the problem associated with test case prioritization requires understanding which is defined as follows:

Given:

T is test suites based on requirements, PT refers to a number of ways they are chosen, where f is a function whose value depends on permutation of these T to some real number

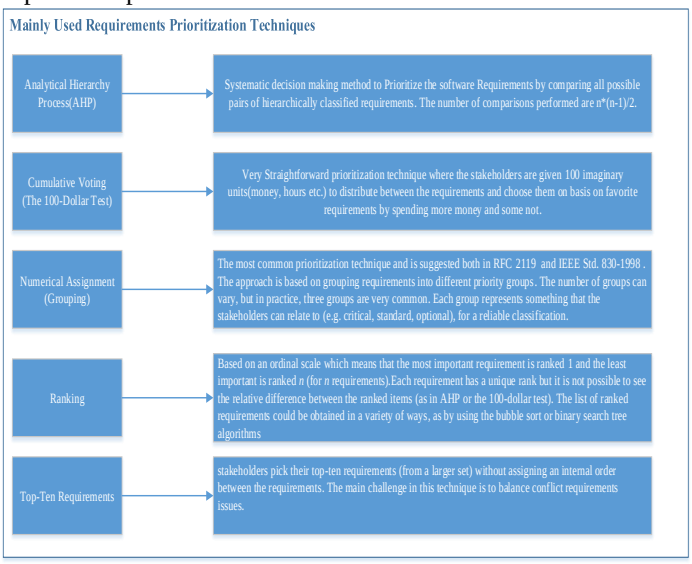

Fig.2:-Some mainly Requirements Prioritization Techniques

Problem:
We have to find T'
Such that T' ϵ PT For all T, (T ' ϵ PT)
Where (T! = T') and f (T') >= f (T).

IV. APPROACH FOR TEST CASE PRIORITIZATION TECHNIQUE

Existing algorithms for test case prioritization are based on greedy approach. Greedy algorithm selects maximum weighted element based on the criteria decided to be chosen. It can be statement coverage, branch coverage, function coverage and fault coverage. Some of the techniques are mentioned in table I. The same procedure is followed until gets the order of test cases with suboptimal solution. Elements are sorted in an increasing order using quick sorting which gives complexity of O (mn) where m is the no. of statements and n is no. of test cases. The major drawback of greedy algorithm is that it gives a local optimum solution of the problem concern. The solution provided could be either maximal or minimal based on neighboring available test cases. Next to the simple greedy algorithm, additional greedy algorithm came into existence which is similar as Greedy algorithm with the use of different strategies. The test cases that focus on maximum coverage are selected by using greedy algorithm and the information of previous covered code additionally used to perform greedy algorithm which gives complexity of order 0 (mn2) . Another 2-0ptimal greedy algorithm used which is based on the travelling salesmen problem i.e. "finding the minimum cost path passing through every node in a graph G at least once". As per the statement test cases, achieving complete coverage earlier has given highest priority. The pair of test case is selected with some readjustment of coverage information which gives complexity of O(mn3). 2-0ptimal algorithm and additional greedy algorithm are little similar. Popularize met heuristic search techniques help in finding a solution to a certain problem with reasonable computational cost. It implements some sort of stochastic optimization in which the resultant solution is dependent on the number of random variables generated. Hill climbing and Genetic algorithm comes under such technique. Hill climbing algorithm is very simple to implement. In this algorithm, the initial solution state is randomly selected which will further compare with its neighboring states. If the neighboring state has higher fitness then it will become the current state and thus the most appropriate state has been chosen to get a solution state. State here, refers to the test suite contains test cases and neighbors are the same test suite with different ordering of test cases.

RCis the value (1 to 10) assigned by the developer based on the perceived implementation difficulties of the requirement. RV is the number of times a requirement has changed. Higher factor values indicate a need for prioritization of test case related to that requirement. Based on the project and customer needs, the development team assigns weight to the PFs such that the assigned total weight (1.0) is divided amongst the PFV.For every requirement, Equation 1 is used to calculate a weighted prioritization (WP) factor that measures the importance of testing a requirement earlier. Test cases are then ordered such that the test cases for requirements with high WP are executed before others.
Prioritization Factor Value (pfv) = PFvalue * PFweight

V. DISCUSSION AND ALGORITHM

GAPRIORITIZE (P,T,s,gmax,pc,pm,pa,Pd,Tc,w)
Input: program P
Test suit T
Number of tuples to be created per iterations s
Percent of total requirement coverage pr
Crossover probability :- pc
Mutation probability :-pm
Addition probability:- pa
Deletion probability :-pd
Test adequacy criteria :-tc
Program coverage weight w, based on numbers of prioritized requirement covered.
Output : Maximum fitness tuple Fmax ε F in set Rmax
tmax<- total number of requirements.
R0<-null;
repeat
R0<-R0 U {random individual created}
until |R0|=s
g<-0
repeat
F<-0
For Rj ε Rg
F<-FUCalculateFitness(P,Rj,pt,tc,w)
F<-fv*fw;
Rg+1 <- SelectTwoTuple(Rg,F).
repeat
Rk,Rl<-SelectParents(Rg,F)
Rq,Rr<-ApplyCrossover(pc,Rk,Rl)
Rq<-ApplyMutation(pm,Rq)
Rq<-ApplyMutation(pm,Rq)
Rq<-AddAdditionalTests(T,pa,Rq)
Rr<-AddAdditionalTests(T,pa,Rr)
Rq<-DeleteATest(pd,Rq)
Rr<-DeleteATest(pd,Rr)
Rg+1<-Rg+1U{Rq}U{Rr}
until |Rg+1|=s
g<-g+1
until g>gmax
Rmax<-FindMaxFitnessTuple(Rg-1,F)
returnRmax

A genetic algorithm is used to solve the problem. First execution of each test case is recorded to find that the particular test case is covering which and how many requirements.First find the total number of requirements for the system and store

them in tmax. Select the tuples that is covering maximumnumber of requirements and make a set sayR0.
Apply the fitness function to find each of Rj and its fitnessvalue is denoted by Fj to determines its quality. Select the two best tuple using R0 and F to form Rg which will be the next generation. Identifies Rk and Rl through a roulette wheel selection technique based on the probability proportional to |F|.the fitness values are normalized in relation to the rest ofthe test tuple set by multiplying each FjεF by a fixed number,so that sum of all fitness values equals one. The test tuples are then sorted by descending fitness values, and the accumulated fitness values are calculated.

A random number r ε [0,1] is next generated, and the first individual whose accumulatednormalized value is greater than orequal to r is selected. This selection method is repeated until enough tuple are selected to fill the set Rg. Then apply crossover that may merge the pair {Rk,Rl} to create the two potentially new tuples {Rq,Rr} based on pc, a user provided crossover probability. Each tuple in the pair {Rq,Rr} may then be muted based on pm, a user provided mutation probability. Finally a new test case may be added or deleted from Rq and Rr. After each of these modifications have been made to the original pair, both tuples Rq and Rr are entered into R1. The same transformation are applied to allpairs selected by R0 until R1contains s test tuples, In total gmax sets of s test tuples are iteratively created in this fashion. When the final set Rgmax has been created, the test tuple with the greatest fitness, Rmax, is determined. This tuple is guaranteed to be the tuple with the highest fitness out of allg sets of size s.

VI. CONCLUSION

Fitness function for this algorithm would be calculated in the following form in which fitness function has two components. The first component is primary fitness which we calculated in this form:
Fpri(P,Ri, tc,w) = rc(P,Ri, tc) * w, where Fpri=primary fitness , andthe second component would take the following form:
Fsec(P,Ri, tc) = Fs-actual(P,Ri, tc)/Fs-max(P,Ri, tc), where
Fsec=second component considers the incremental
requirement coverage of the tuple, giving precedence to test tuples whose earlier tests have greater coverage. Fsec is also calculated in two parts.
(i) Fs-actual-> calculated by summing the product of the requirement coverage and fault severity of faults generated by respective test cases.
(ii) F-actual->that represent the maximum value that Fsactual can take.

As an example of a fitness calculation, let the program coverage weight w = 100, P be a program, and tc be a test adequacycriterion (e.g., requirement coverage). Suppose Ri=<T1,T2,T3>. Also, assume we have severity information based on requirement prioritization is T1=5,T2=3, T3=2, and test tuple requirement coveragerc(P,Ri, tc) = 0.20.

Then, primary fitness =0.2*100=20

Fsec next gives preference to test tuples that have more high prioritized requirement covered early in execution. To calculate Fsec, the requirement coverages of Ri(1,1)=<T1>,Ri(1,2)=<T1,T2>,Ri(1,3)=<T1,T2,T3> must be measured. Suppose for this example that rc(P,Ri(1,1),tc) = 0.05, rc(P,Ri(1,2),tc) = 0.19, and, as already known, rc(P,Ri(1,3), tc) = rc(P,Ri, tc) =0.20. Fsec is calculated as follows,
Fs-actual(P,Ri, tc) = (5* 0.05) + (3* 0.19) + (2* 0.20)= 1.27
Fs-max(P,Ri, tc) = 0.2(5 + 3 + 2) = 2.0,
Fsec(P,Ri, tc) =1.27/2.0= 0.635
Evaluation of this approach:

Consider an example table for understanding the concept of this proposed algorithm. In this case we are using severity value which will be based on requirement weight and value. As the test case is covering maximum number of highly prioritized requirements means it is finding most severe faults early in the testing process.
Issues for GA Practitioners:-

- Choosing basic implementation issues:
  - representation
  - population size, mutation rate, ...
  - selection, deletion policies
  - crossover, mutation operators
- Termination Criteria.
- Performance, scalability.

Solution is only as good as the evaluation function (often hardest part).
Benefits of Genetic Algorithms:-

- Concept is easy to understand.
- Modular, separate from application.
- Supports multi-objective optimization.
- Good for "noisy" environments.
- Always an answer; answer gets better with time.
- Inherently parallel; easily distributed.
- Many ways to speed up and improve a GA-based application as knowledge about problem domain is gained.
- Easy to exploit previous or alternate solutions.
- Flexible building blocks for hybrid applications.
- Substantial history and range of use.

When to Use a GA:-

- Alternate solutions are too slow or overly complicated.
- Need an exploratory tool to examine new approaches.
- Problem is similar to one that has already been successfully solved by using a GA.
- Want to hybridize with an existing solution.
- Benefits of the GA technology meet key problem Requirements.

Approaches and Challenges
Possible approaches:-

- Case Studies to find out requirement factors and optimize them using GA.

Challenges are:-

- Find test cases with the greatest fitness to prioritize test case.(Implementation of fitness function using GA based on requirement factors).

Objective

- Detection of severe faults in less cost and less time to reduce cost and time by optimization of regression test selection.

Techniques

- by using Requirement prioritization considering Business value.
- Selection: Code-coverage TCP techniques using requirement factors.
- Prioritization : Prioritization of Requirements for Testing (PORT Version 1.1) and Genetic Algorithm.

Tool used

Mat Lab used for implementation and evolution of the required fittest function based on requirement factors provide by user/stakeholder.

VII. REFERENCES

[1] K. F. Man, Member, IEEE, K. S. Tang, and S. Kwong, Member, IEEE"Genetic Algorithms: Concepts and Applications," IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS, VOL. 43, NO. 5, OCTOBER 1996.

[2] G. Rothermel, R.H. Untch, C. Chu, and M.J. Harrold, "Test Case Prioritization," Technical Report GIT-9-28, college of computing ,Georgia institute of technology ,Dec.,1999.

[3] Paolo Tonella, Angelo Susi, Francis Palma, "Using Interactive GA for Requirements Prioritization," 2nd International Symposium on Search Based Software Engineering.

[4] HemaSrikanth, Laurie Williams, Jason Osborne, "System test case prioritization of new and regression test cases," Proceedings of the seventh international workshop on Economics driven software engineering research, pages 64-73, May 2005 IEEE.

[5] Dr. Varun Kumar, Sujata, Mohit Kumar, "Test Case Prioritization Using Fault Severity," IJCST Vol. 1, Issue 1, September 2010.

[6] G. Rothermel, R. Untch, C. Chu, and M. Harrold,"Test Case Prioritization," IEEE Transactions on Software Engineering, vol. 27, pp. 929-948, October 2001.

[7] S. Elbaum, A. Malishevsky, and G. Rothermel,"Test Case Prioritization: A Family of Empirical Studies,"IEEE Trans. on Software Engineering, vol. 28, February2002.

[8] Md. ImrulKayes," Test Case Prioritization for Regression Testing Based on Fault Dependency", (2011) IEEE.

[9] Zheng Li, Mark Harman, and Robert M. Hierons, "Search algorithms for regression test case prioritization," IEEE Trans. On Software Engineering, vol 33, no.4, April 2007 .

[10] Wang Jun, Zhuang Yan, Jianyun Chen," Test Case Prioritization Technique based on Genetic Algorithm", (2011) IEEE.

[11] Kristen R. Walcott Mary Lou Soffa, Gregory M. Kapfhammer Robert S. Roos, "Time Aware Test Suite Prioritization," ISSTA'06, July 17–20, (2006), Portland, Maine, USA [12.] G. Duggal, B. Suri ,"Understanding Regression Testing Techniques", COIT, (2008), India.

[12] Dr. Varun Kumar, Sujata, Mohit Kumar, "Requirment based Test casse Prioritization using Genetic Algorithm" ,IJCST Vol. 1, Issue 1, December, 2010.

[13] Thillaikarasi Muthusamy1 and Dr. Seetharaman, "EFFECTIVENESS OF TEST CASE PRIORITIZATION TECHNIQUES BASED ON REGRESSION TESTINGInternational Journal of Software Engineering & Applications (IJSEA), Vol.5, No.6, November 2014.

[14] G.N. Purohit, Sujata, "A Schema Support for Selection of Test Case Prioritization Techniques," International Journal of Software Engineering & Applications (IJSEA), Vol.5, No.6, November 2014.

[15] Ms. Sujata, and Nancy Dhamija,"Test Cases Prioritization Using Model Based Test Dependencies: A survey,"International Journal of Information & Computation Technology.ISSN 0974-2239 Volume 4, Number 10 (2014), pp. 1003-1010.

[16] PatrikBerander and AnnelieseAndrews,"RequirementsPrioritization,"inengneering and managing software reqquirements,edited by A. Aurum and C. Wohlin,springerVerlag.

[17] Neha Sharma, Sujata, Prof. G.N. Purohit Test Case Prioritization Techniques HAn Empirical Study"", 2014 IEEE.

[18] Sujata, Mohit Kumar, Dr. Varun Kumar ,"Requirements based Test Case Prioritizationusing Genetic Algorithm," IJCST Vol. 1, Iss ue 2, December 2010.

[19] Jyoti1, Mrs. LekhaBhambhu, "An Efficient Genetic Algorithm for Fault Based Regression Test Case Prioritization," International Journal of Advanced Research in Computer and Communication Engineering Vol. 4, Issue 8, August 2015.

[20]]https://www.google.co.in/