Clustering Based Prioritization of Test Suites in Software Testing

Arvind Kumar upadhyay[1] and Dr. A. K. Misra[2]

1. Deptt. of Computer Sc. and Engg. EIT Faridabad (HR)-India
E.mail:aupadhayay1@gmail.com
2. Deptt. of Computer Sc. and Engg, MNNIT Allahabad (UP)-India

Abstract

Software Testing challenges the development process at all stages of software development and it is no surprise that a bulk of development cost is spent on testing. Testing is fundamental to software quality and is ultimate review of specification, design and coding. Human fallibilities are enormous and errors may begin to occur at the very inception of ideas. The focus of Test case design is on a set of techniques that meet overall testing objectives. New test cases are a necessity in this ever evolving scenario of software development. Many a times the size of a test suite may become so large that it becomes necessary to apply some control mechanism on these numbers of test cases. Prioritization is a technique that can facilitate increased chances of early fault detection and is helpful in reducing test suite size. In our attempt we adopt a new technique CBP (clustering based prioritization) to effectively control test suite size.

I. Introduction

Due to the ever changing target requirements, the test suites continue to grow and there may be circumstances when obsolete and redundant test cases entail necessary attention. It is always advantageous to have a small set of test cases to avoid repeated execution of tests cases. Through minimizations, redundant and obsolete tests cases could be eliminated [1]. Software cost is size dependant. Test suite minimization technique can lower cost by reducing a test suite to a minimal subset. By reducing test suite, maintenance cost is significantly minimized. Prioritization is fundamental to test suite minimization process [7]. The rationale behind prioritization is to reduce test cases based on some non arbitrary criteria and always aiming to select the most appropriate tests. For instance following priority categories may be determined for the test cases:

Priority 1. The test cases must be executed before the final product is released to remove the critical bugs.
Priority 2. If time permits, the test cases may be executed.
Priority 3. The test cases are not important prior to the current release. It may be tested shortly after the release of the current software version.
Priority 4. The test case is never important, as its impact is nearly negligible.
Such a priority scheme ensures that low priority test cases do not create problems for software[9]. At times customers demand that some important features of software be tested and presented in the first version of software itself. There important features become criteria. Priority can be advertisement based because the company might have promised about essential features to customers[5]. Fault detection rate of a test suite reveals about the likelihood of faults earlier. Coverage criteria should be met earlier in test process.

2. Test case prioritization

The purpose of Test case prioritization lies in ordering test cases based on a particular technique [21]. It takes into account that if such a scheme is followed then it is more likely to meet the objective than it would otherwise. Test case prioritization can address a wide variety of objectives as:

1. To increase the rate of fault detection so that faults may be revealed earlier in regression test.
2. To focus on high-risk faults and detect them earlier in testing process.
3. To speed up the regression errors connected to code changes as early as possible.
4. To cover code coverage in the system under test at a faster rate.
5. To enhance reliability confidence in the system under test at a faster rate.

3. Clustering based prioritization

3.1 Motivation

The total number of comparisons required for pair-wise comparison is $O(n^2)$ comparisons[20]. Redundancy makes pair-wise comparison very robust but the high cost incurred discourages it from being applied to test case prioritization. The maximum number of comparisons a human can make consistently is approximately 100 [1]; above this threshold, inconsistency grows significantly, leading to reduced effectiveness. But to require less than 100 pair-wise comparisons, the test suite could contain no more than 14 test cases. In real world scenario the issue of scalability is challenging. For example, suppose there are 1,000 test cases to prioritize; the total number of required pair-wise comparisons would be 499,500. Obviously it is unrealistic to expect a human tester to provide reliable responses for such a large number of comparisons [8]. Our approach using K-means cluster based prioritization reduces the number of comparisons and can be very effective. Instead of prioritizing individual test cases, clusters of test cases are prioritized using techniques such as CBP.

3.2 K-means clustering criteria

Broadly speaking, there are two methods of clustering i.e. data can be arranged as a group of individuals or as a hierarchy of

groups. It can thereafter be established that whether the data group belong to some preconceived ideas or suggest new ones[4]. Cluster analysis groups data objects into clusters such that objects belonging to the same cluster are similar, while those belonging to different ones are dissimilar. Clustering techniques could be categorized into modes Partitional or Hierarchical:

Partitional: Given a database of objects, a partitional clustering algorithm constructs partitions of the data, where each cluster optimizes a clustering criterion, such as the minimization of the *sum of squared distance from the mean* within each cluster[6]. The complexity of Partitional clustering is large because it enumerates all possible groupings and tries to find the global optimum. Even for a small number of objects, the number of partitions is huge. That's why; common solutions start with an initial, usually random, partition and proceed with its refinement. A better practice would be to run the partitional algorithm for different sets of initial points (considered as representatives) and investigate whether all solutions lead to the same final partition[13]. Partitional Clustering algorithms try to locally improve a certain criterion. First, they compute the values of the similarity or distance, they order the results, and pick the one that optimizes the criterion[11]. Hence, the majority of them could be considered as greedy-like algorithms.

Hierarchical: Hierarchical algorithms create a hierarchical decomposition of the objects. They are either *agglomerative* (*bottom-up*) or *divisive* (*top-down*):

(a) *Agglomerative* algorithms start with each object being a separate cluster itself, and successively merge groups according to a distance measure[14]. The clustering may stop when all objects are in a single group or at any other point the user wants. These methods generally follow a greedy-like bottom-up merging.

(b) *Divisive* algorithms follow the opposite strategy[12]. They start with one group of all objects and successively split groups into smaller ones, until each object falls in one cluster, or as desired[10]. Divisive approaches divide the data objects in disjoint groups at every step, and follow the same pattern until all objects fall into a separate cluster. This is similar to the approach followed by divide-and-conquer algorithms.

K-means clustering method:

K-means *clustering* methods produce clusters from a set of objects based upon the squared-error objective functions:

$$E = \sum_{i=1}^{k} \Sigma_{p \in C_i} |p - m_i|^2$$

being minimized[2,3]. In the above expression, c_i are the clusters, p is a point in a cluster c_i and m_i the mean of cluster c_i. The mean of a cluster is given by a vector, which contains, for each attribute, the mean values of the

data objects in this cluster, input parameter is the number of clusters, k[22]. As an output the algorithm returns the centers, or means, of every cluster c_i, most of the times excluding the cluster identities of individual points. The distance measure usually employed is the Euclidean distance. Both for the optimization criterion and the proximity index, there are no restrictions, and they can be specified according to the application or the user's preference. The algorithm is as follows:

1. Select k objects as initial centers;
2. Assign each data object to the closest center;
3. Recalculate the centers of each cluster;
4. Repeat steps 2 and 3 until distribution of data objects in clusters do not change;

The algorithm is relatively scalable.

4. The Experiment

4.1. Research Questions

We are interested in the following research question.

Q**:** How can K-means clustering technique facilitate test case prioritization of test suites?

4.2. Efficacy and CBP Measures

In his classic book, Glenford Myers proposes the following testing problem: Develop a good set of test cases for a program that accepts three numbers, a, b, and c, interprets those numbers as the lengths of the sides of a triangle, and outputs the type of the triangle[18,19]. For this classic triangle problem, we can divide the domain space into three sub domains, one for each different type of triangle that we will consider: scalene (no sides equal), isosceles (two sides equal), and equilateral (all sides equal). We can also identify two error situations: a sub domain with bad inputs and a sub domain where the sides of those lengths would not form a triangle. Additionally, since the order of the sides is not specified, all combinations should be tried. Finally, each test case needs to specify the value of the output. In the following example, we show some typical values and conditions to decide triangle formation [23].

Sub domain Example 1: Test Cases for Triangle formation
Scalene:
Increasing size (3, 4, 5—scalene)
Decreasing size (5, 4, 3—scalene)
Largest as second (4, 5, 3—scalene)
Isosceles:
a=b & other side larger (5, 5, 8—isosceles)
a=c & other side larger (5, 8, 5—isosceles)
b=c & other side larger (8, 5, 5—isosceles)
a=b & other side smaller (8, 8, 5—isosceles)
a=c & other side smaller (8, 5, 8—isosceles)
b=c & other side smaller (5, 8, 8—isosceles)
Equilateral:
All sides equal (5, 5, and 5— equilateral)
Not a triangle:
Largest first (6, 4, 2—not a triangle)
Largest second (4, 6, 2—not a triangle)
Largest third (1, 2, 3—not a triangle)
Bad inputs:
One bad input (-1, 2, 4—bad inputs)
Two bad inputs (3,-2,-5—bad inputs)
Three bad inputs (0, 0, 0 – bad inputs)

This list of sub domains could be increased to distinguish other sub domains that might be considered significant. For example, in scalene sub domains, there are actually six different orderings, but the placement of the largest might be the most significant based on possible mistakes in programming.

STRUCTURAL TESTING

Structural testing is based on the structure of the source code. The simplest structural testing criterion is every statement coverage, often called C0 coverage[15,17]. This criterion is that every statement of the source code should be executed by some test case at least once. The normal approach to achieving C0 coverage is to select test cases until a coverage tool indicates that all statements in the code have been executed. In the following pseudo code implementation of the triangle problem, the matrix shows which lines are executed by which test cases. Note that the first three statements (A, B, and C) can be considered parts of the same node.

Table 1

Node	Source Line	3,4,5	3,5,3	0,1,0	4,4,4
A	read a,b,c	*	*	*	*
B	type=''scalene''	*	*	*	*
C	if(a==b||b==c||a==c)	*	*	*	*
D	type=''isosceles''		*	*	*
E	if(a==b&&b==c)	*	*	*	*
F	type=''equilateral''				*
G	if(a>=b+c||b>=a+c||c>=a+b)	*	*	*	*
H	type=''not a triangle''			*	
I	if(a<=0||b<=0||c<=0)	*	*	*	*
J	type=''bad inputs''			*	
K	print type	*	*	*	*

Control flow graph consists of nodes and edges. For our example of triangle formation problem, the control flow graph can be drawn as below:

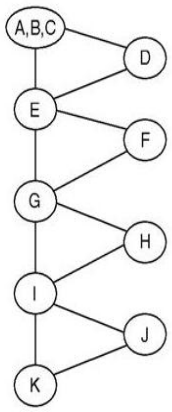

fig1: Control flow graph for example 1.

If we model the program of Example 1, as a control flow graph, then coverage criterion requires covering every arc in the control flow diagram[16]. A path is a unique sequence of program nodes that are executed by a test case. In the testing matrix (Table1) above, there were eight sub domains. Each of these just happens to be a path. The following table 2 shows the eight feasible paths in the triangle pseudo code of Example 1.

Table2

Path	T/F	Test Case	Output
ABCEGIK	FFFF	3,4,5	Scalene
ABCEGHIK	FFTF	3,4,8	Not a triangle
ABCEGHIJK	FFTT	0,5,6	Bad inputs
ABCDEGIK	TFFF	5,8,5	Isosceles
ABCDEGHIK	TFTF	3,8,3	Not a triangle
ABCDEGHIJK	TFTT	0,4,0	Bad inputs
ABCDEFGIK	TTFF	3,3,3	Equilateral
ABCDEFGHIJK	TTTT	0,0,0	Bad inputs

5. Result & Analysis:

We now apply the k-means clustering method for triangle problem. For this we make use of table 2. In table 2 there are eight paths in every path testing criteria. Initially we took three clusters as k-value and by using the algorithm we finally calculate that three clusters have following combination:

C1: path1

C2: path2, path3, path4, path5, path7

C3: path6, path8

In our prioritization, we priorities the clusters having maximum numbers of paths in descending order to execute test cases. So for our example path2, path3, path4, path5, path7 would be executed first as this has got highest priority. In the second step path6, path8 would be tested. In the third step path1 would be executed. Now it can be seen that cluster 2 is most important, cost effective yet complete.

In the following figure, we have shown dendrogram to show the prioritization of test cases.

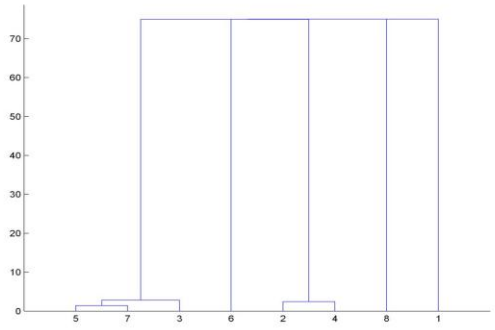

Figure 2: Dendrogram of test cases

Out of all test cases through 1 to 8, the dendrograph shows that the priority of test cases are 5,7,3,6,2,4,8,1. It also shows that test cases 5 and 7 are highest priority combination and the combination of two along with 3 is a must in terms of testing. There after the 2 and 4 combination is equally important combination as far testing is concerned.

6. Conclusion & Future Work

In our paper we make use of clustering technique as prioritization measure to priorities the test cases. Our approach can categories the test cases in decreasing order of importance of test cases and thus save vital time and cost of software development. In future, however we would like to take it forward and use various data mining techniques to find association among test suites and cost and space occupied and thus predict the most suitable test cases vs. the requirements. In a way we can successfully link software development with user requirements.

7. **References:**

[1] Gregg Rothermel ,Roland H.Untch,Mary Jean Harrold,"Prioritizing Test Cases For Regression Testing," IEEE Transaction on Software Engineering, Vol.27, No.10 October 2001.

[2] G. J. Myers. The Art of Software Testing. Revised and Updated by Tom Badgett and Todd M.Thomas with Corey Sandler, John Wiley & Sons, Inc, Second Edition. 2004, pp. 1-255.

[3] Boris Beizer. *Software System Testing and Quality Assurance*. Van Nostrand, New York, 1984.

[4] Boris Beizer. *Software Testing Techniques*. Van Nostrand Reinhold, Inc, New York NY, 2nd edition, 1990. ISBN 0-442-20672-0.

[5] Richard A. DeMillo, W. Michael McCracken, Rhonda J. Martin, and John F. Passafiume.*Software Testing and Evaluation*.Benjamin/Cummings, Menlo Park CA, 1987.

[6] Siripong Roongruangsuwan, Jirapun Daengdej,"Test Case prioritization techniques," Journal of theoretical and applied informational technology,2005

[7] Mao ye, boqinFeng, yao Lin 7Li Zhu. "Neural Networks Based Test Case Selection" Proc of IEEEtransactions,2006.

[8] T.Y. Chen, Pak-lok poon, t.h. Tse."A choice Relation framework for supporting Category-partition Test Case generation" IEEE transactions on software Engineering, vol.29, No.7, July 2003.

[9] Sebastian Elbaum, Alexey G.Malishevsky, Gregg Rothermel."Test Case Prioritization" IEEE transactions on software Engineering, vol.28, No.2, February 2002.

[10] Kuo –Chung Tainand Yu Lei. "A Test generation strategy for Pairwisetesting" IEEE transactions on software Engineering, vol.28, No.1, January 2002.

[11] Christoph C. Michael, gary McGraw, Michael A. Schatz. "Generating software test data by Evolution". IEEE transactions on software Engineering, vol.27, No.12, December 2001.

[12] Shino yahoo & Mark Harman ,Paolo tonella & Angelo susi, "Clustering test cases to achieve Effective & scalable prioritisation incorporating Expert knowledge," ISSTA 09,July 19-23, 2009, Chicago, USA.

[13] Gregg rothermel , roland h. untch, chengyun chu ,mary jean harrold, " Test case prioritization : An Empirical study," Proceedings of the international conference on software maintenance, oxford ,U.K., September ,1999, IEEE

[14] Wei-Tek Tsai and Lian Yu, Feng Zhu, Ray Paul. "Rapid embedded system testing using verification patterns" . IEEE software 2005.

[15] S. G. Elbaum, A. G. Malishevsky, and G. Rothermel. Prioritizing test cases for regression testing. In International Symposium on Software Testing and Analysis, pages 102–112. ACM Press, 2000.

[16] Martina marre and Antonia Bertolino, "using spanning sets for coverage testing". IEEE transactions on software Engineering, vol.29, No.11, November 2003.

[17] H. Do, S. G. Elbaum, and G. Rothermel. Supporting controlled experimentation with testing techniques: an infrastructure and its potential impact. *Empirical Software Engineering*, 10 (4): 405–435, 2005.

[18] H. Do, G. Rothermel, and A. Kinneer. Prioritizing JUnit Test cases: an empirical assessment and cost-benefits analysis.*Empirical Software Engineering*, 11: 33–70, 2006.

[19] P. M. Duvall, S. Matyas, and A. Glover. *Continuous Integration: Improving Software Quality and Reducing Risk*. Addison Wesley, Upper Saddle River, NJ, 2007.

[20] S. G. Elbaum, A. G. Malishevsky, and G. Rothermel. Prioritizing test cases for regression testing. *ACM SIGSOFT Software Engineering Notes*, 25 (5): 102–112, 2000.

[21] S. G. Elbaum, A. G. Malishevsky, and G. Rothermel. Test

case prioritization: a family of empirical studies. *IEEE Transactions on Software Engineering*, 28 (2): 159–182, 2002.

[22] S. G. Elbaum, G. Rothermel, S. Kanduri, and A. G. Malishevsky. Selecting a cost-effective test case Prioritization technique. *Software Quality Control*, 12 (3): 185–210, 2004.

[23] David Gustafson,"Theory and Problem of Software Egineering," Computing and Information science Department Kansas state University

.