Knowledge Mining of Test Case System

Lilly Raamesh/ Research Scholar
C.S.E Department
Anna University Chennai
Chennai, Tamil Nadu, India
lillyraamesh@yahoo.co.in

G.V. Uma/Asst. Prof
C.S.E Department
Anna University Chennai
Chennai, Tamil Nadu, India

Abstract — The paper analyzes knowledge mining of the test case System. Widespread use of test case systems and explosive growth of databases require traditional manual data analysis to be coupled with methods for efficient computer-assisted analysis It is very important for us to utilize this kind of information effectively. Today, the design of test case with enhanced reliability is a real challenge as it needs expert designers with perfect knowledge about the whole system and also the traditional test case generation approach faces a challenge in test results analysis, because of the difference between the generated test cases and the expected results. So, it will be better if we could establish a method for automatic test case mining based on both program structures and the functional requirements in specifications. Mining approach can be used to have a perfect knowledge about the whole system. In order to improve the accuracy and efficiency of knowledge acquisition, it establishes a knowledge mining model.

Key words – data mining, test case, test suite, knowledge mining, clustering.

I. INTRODUCTION

Data mining sits at the interface between statistics, computer science, artificial intelligence, machine learning, database management and data visualization. It is the process of identifying valid, novel, potentially useful, and ultimately comprehensible knowledge from data that is used to help by crucial decision-making. The search for an optimal solution in the test case generation problem has a great computational cost and for this reason these techniques try to obtain near optimal solutions. As a consequence, they have attracted growing interest from many researchers in recent years. So, we are trying to apply mining techniques.

II. TEST CASE

A test case has components that describe an input, action or event and an expected response, to determine if a feature of an application is working correctly. The basic objective of writing test cases is to validate the testing coverage of the application. So writing test cases brings some sort of standardization and minimizes the ad-hoc approach in testing.

A test case is usually a single step, or occasionally a sequence of steps, to test the correct behaviour/functionalities, features of an application. An expected result or expected outcome is usually given.

Additional information that may be included:

- test case ID
- test case description
- test step or order of execution number
- related requirement(s)
- depth
- test category
- author
- check boxes for whether the test is automatable and has been automated.

Additional fields that may be included and completed when the tests are executed:

- pass/fail
- remarks

Larger test cases may also contain prerequisite states or steps, and descriptions.

III. TEST SUITE

A test suite is a collection of test cases that are intended to be used to test a software program to show that it has some specified set of behaviours. A test suite often contains detailed instructions or goals for each collection of test cases and information on the system configuration to be used during testing. A group of test cases may also contain prerequisite states or steps, and descriptions of the tests.

IV. IMPORTANCE OF TEST CASES

Writing effective test cases is a skill and that can be achieved by some experience and in-depth study of the application on which test cases are being written.

Designing good test cases is a complex art. The complexity comes from three sources:

- Test cases help us discover information. Different types of tests are more effective for different classes of information.
- Test cases can be “good” in a variety of ways. No test case will be good in all of them.
- People tend to create test cases according to certain testing styles, such as domain testing or risk-based testing. Good domain tests are different from good risk-based tests.

There are levels in which each test case will fall in order to avoid duplication efforts.

Level 1: In this level we will write the basic test cases from the available specification and user documentation.

Level 2: This is the practical stage in which writing test cases depend on actual functional and system flow of the application.

Level 3: This is the stage in which we will group some test cases and write a test procedure. Test procedure is nothing but a group of small test cases maximum of 10.

Level 4: Automation of the project.

So we can observe a systematic growth from no testable item to an Automation suite.

V. SOFTWARE TESTING

Software testing is the process of executing a program in order to find faults, thus helping developers to improve the quality of the product when the discovered faults are solved and reducing the cost produced by these faults. A software test consists of a set of test cases, each of which is made up of the input of the program, called test data, and the output that must be obtained. As the target of software testing is to find faults, a test is successful if an error is found.

Testing is a very important, though expensive phase in software development and maintenance; A challenging part of this phase entails the generation of test cases. This generation is crucial to the success of the test, because it is impossible to achieve a fully tested program given that the number of test cases needed for fully testing a software program is infinite, and a suitable design of test cases will be able to detect a great number of faults.

VI. AUTOMATED TESTING

High-volume automated testing involves massive numbers of tests, comparing the results

Another approach runs an arbitrarily long random sequence of regression tests. Tests that the program has shown it can pass one by one. Memory leaks, stack corruption, wild pointers or other garbage that cumulates over time finally causes failures in these long sequences.

Yet another approach attacks the program with long sequences of activity and uses probes (tests built into the program that log warning or failure messages in response to unexpected conditions) to expose problems.

High-volume testing is a diverse grouping. The essence of it is that the structure of this type of testing is designed by a person, but the individual test cases are developed, executed, and interpreted by the computer, which flags suspected failures for human review. The almost complete automation is what makes it possible to run so many tests.

The individual tests are often weak. They make up for low power with massive numbers.

Because the tests are not handcrafted, some tests that expose failures may not be particularly credible or motivating. A skilled tester often works with a failure to imagine a broader or more significant range of circumstances under which the failure might arise, and then craft a test to prove it.

Some high-volume test approaches yield failures that are very hard to troubleshoot. It is easy to see that the failure occurred in a given test, but one of the necessary conditions that led to the failure might have been set up thousands of tests before the one that actually failed. Building troubleshooting support into these tests is a design challenge that some test groups have tackled more effectively than others.

Testers continually learn about the software they're testing, the market for the product, the various ways in which the product could fail, the weaknesses of the product (including where problems have been found in the application historically and which developers tend to make which kinds of errors), and the best ways to test the software. At the same time that they're doing all this learning, exploratory testers also test the software, report the problems they find, advocate for the problems they found to be fixed, and develop new tests based on the information they've obtained so far in their learning."

An exploratory tester might use any type of test--domain, specification-based, stress, risk-based, any of them. The underlying issue is not what style of testing is best but what is most likely to reveal the information the tester is looking for at the moment.

Exploratory testing is not purely spontaneous. The tester might do extensive research, such as studying competitive products, failure histories of this and analogous products, interviewing programmers and users, reading specifications, and working with the product.

What distinguishes skilled exploratory testing from other approaches and from unskilled exploration, is that in the moments of doing the testing, the person who is doing exploratory testing well is fully engaged in the work, learning and planning as well as running the tests. Test cases are good to the extent that they advance the tester's knowledge in the direction of his information-seeking goal. Exploratory testing is highly goal-driven, but the goal may change quickly as the tester gains new knowledge.

Here, we prefer exploratory testing. For that we choose mining of test cases method, Since mining of test cases provides the complete knowledge about the total system.

VII. AUTOMATIC GENERATION OF TEST CASES

The techniques for the automatic generation of test cases try to efficiently find a small set of cases that allow a given adequacy criterion to be fulfilled, thus contributing to a reduction in the cost of software testing. Model checking techniques can be successfully employed as a test case generation technique to generate tests from formal models.

VIII. TEST SUITE REDUCTION

Test-suite reduction can provide us with a smaller set of test cases that preserve the original coverage-often a dramatically smaller set. One potential drawback with test-suite reduction is that this might affect the quality of the test-suite in terms of fault finding. Using the knowledge mining technique, we reduced the noriginal test suite into smaller set of test cases that preserve the original coverage.

IX. IX. DATA MINING

Data mining aims to find patterns in organizational databases. However, most techniques in mining do not consider knowledge of the quality of the database. In this work, we show how to incorporate clustering and classification mining. Real life data sets are often interspersed with noise, making the subsequent data mining process difficult.

The task of the classifier could be simplified by eliminating attributes that are deemed to be redundant for classification, as the retention of only pertinent attributes would reduce the size of the dataset.

Using a correlation measure between attributes as a fitness measure to replace the weaker members in the population with newly formed chromosomes makes improvements.

X. KNOWLEDGE MINING OF THE TEST CASE SYSTEM

In order to evaluate the effect of knowledge mining of the Test Case System, good or bad, we must build a relative knowledge evaluation system. The result can objectively indicate the value created by Test Case.

A. The accuracy of the knowledge

Reasonable knowledge discovery algorithm: This is the main part of the Test Case data mining algorithms.

B. The understandability of knowledge

Knowledge representation and explain mechanism: It is to evaluate the function of Test Case mining from the user's point of view. More easily the knowledge to be understood by users, it plays a greater role. So a very important point is user can understand a new knowledge, this requires the knowledge must be explained in a simple way in the system, such as graphics, natural language and visualization technologies. When data mining discovered a new knowledge, it is able to explain it by the forms of relation, rule and concept. But user will not know the basic principles of the find or to distinguish the value of the knowledge until the system provides a better explanation mechanism.

C. The benefits of knowledge

- Direct benefits: the direct benefits created by business intelligence include information becoming merchandise, reducing the labour cost and the stock cost.
- Indirect benefits: the indirect benefits created by business intelligence include the accuracy of decision-making, optimizing the supply cycle, improving the competitive skill of employees and fluency of business information.

D. The innovation of knowledge

The knowledge is discovered from the test case system by applying the mining algorithm.

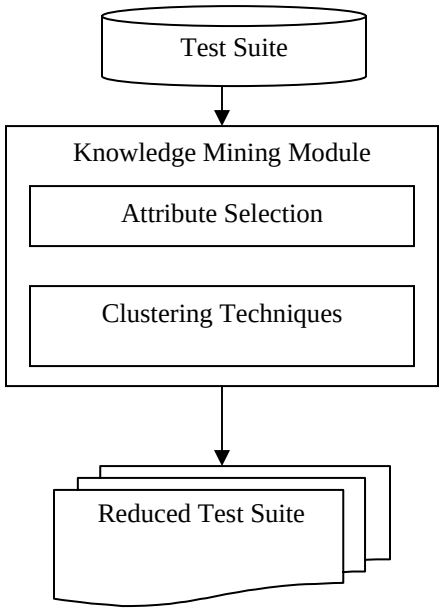


Figure 1. Knowledge Mining of Test Cases

Figure 1. Shows the knowledge mining system for test cases. The test suite is given as the input to the knowledge mining module. This module mines the test cases by attribute selection and by applying the clustering techniques. The output from this module is a reduced test suite.

- Attribute selection in data mining

In the KDD process, interesting patterns and useful relationships are attained from the analysis of the input data. To ensure that the patterns derived are as accurate as possible, it is essential to improve the quality of the datasets in a pre-processing stage. Most real life data sets contain a certain amount of redundant data, which does not contribute significantly to the formation of important relationships. This redundancy not only increases the dimensionality of the data set and slows down the data mining process but also affects the subsequent classification performance. With this in mind, data reduction aims to trim down the quantity of data that is to be analyzed and yet produce almost similar, if not better, results as compared to the original data. More meaningful relationships can also be derived as the superfluous portions are removed. Attribute selection is the process of removing the redundant attributes that are deemed irrelevant to the data mining task.

In addition, a smaller set of attributes also creates less complicated patterns, which are easily comprehensible, and even visualizable by humans.

The attribute selection can be done using the stepwise forward selection and stepwise backward elimination techniques.

In forward selection, the search begins with an empty set and adds attributes with increasing relevance, before terminating at the point when the classification performance declines.

Backward elimination starts with the complete set of attributes and prunes the most irrelevant attribute after each iteration.

Due to the fact that forward selection begins with an empty set, it neglects the interaction between attributes, which may influence the selection process. On the other hand, backward elimination takes into account this interaction because it begins with a complete set of attributes.

However, the analysis from the full set results in a lengthy runtime and may be unfeasible to carry out if the number of attributes is large.

- Clustering technique

Using clustering algorithm we reduced the test suite. Using *k*-means clustering we reduced the test suite. The *k*-means algorithm assigns each point to the cluster whose center (also called centroid) is nearest. The center is the average of all the points in the cluster — that is, its coordinates are the arithmetic mean for each dimension separately over all the points in the cluster.

The algorithm steps are

- Choose the number of clusters, *k*.
- Randomly generate *k* clusters and determine the cluster centers, or directly generate *k* random points as cluster centers.
- Assign each point to the nearest cluster center.
- Recompute the new cluster centers.
- Repeat the two previous steps until some convergence criterion is met (usually that the assignment hasn't changed).

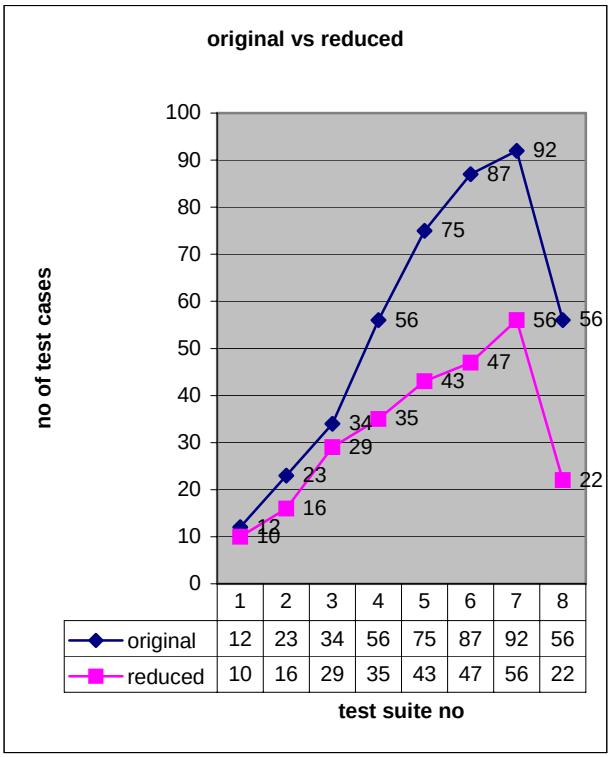
```
//centroid class
import java.io.*;
import java.util.*;
import java.lang.*;
import java.text.*;

public class Centroid {
public void Grouping(double[] Cordx, double[] Cordy, int
clustNumber) {
int clusterNumber = clustNumber;
double[] ClustCordX = new double[clustNumber];
double[] ClustCordY = new double[clustNumber];
this.getMeansetCentroid(Cordx, Cordy, clustNumber);
DecimalFormat dec = new DecimalFormat("0.00");
for(int i = 0;i<Cordx.length;i++) {
String result1 = dec.format(Cordx[i]);
String result2 = dec.format(Cordy[i]);
 }
for(int i = 0; i<clustNumber;i++) {
ClustCordX[i] = Cordx[i];
ClustCordY[i] = Cordy[i];
}
this.groupCordtoCluster(Cordx,Cordy,ClustCordX,ClustCor
dY);
}
public void groupCordtoCluster(double[] Cordx, double[]
Cordy, double[] ClustCordX, double[] ClustCordY) {
double temp ;
int size = Cordx.length;
int clustsize = ClustCordX.length;
int clusterComparison = clustsize;
int[] grouping = new int[size - clustsize];
double[] ClustgroupX = new double[size - clustsize];
double[] ClustgroupY = new double[size - clustsize];
int tempint = -1;
for(int i = clusterComparison; i < size;i++) {
temp = 0;
for(int j = 0;j<clustsize;j++) {
if (j == 0)
tempint++;
if(temp == 0) {
temp = Math.sqrt(Math.pow((Cordx[i]-ClustCordX[j]),2) +
Math.pow((Cordy[i]-ClustCordY[j]),2));
grouping[tempint] = j;
ClustgroupX[tempint] = Cordx[i];
ClustgroupY[tempint] = Cordy[i];
}
else if (temp > Math.sqrt(Math.pow((Cordx[i]-
ClustCordX[j]),2) + Math.pow((Cordy[i]-
ClustCordY[j]),2))) {
temp = Math.sqrt(Math.pow((Cordx[i]-ClustCordX[j]),2) +
Math.pow((Cordy[i]-ClustCordY[j]),2));
grouping[tempint] = j;
ClustgroupX[tempint] = Cordx[i];
ClustgroupY[tempint] = Cordy[i];
}
}
}
DecimalFormat dec = new DecimalFormat("0.00");
String result1, result2, result3, result4;
for(int i = 0; i<grouping.length;i++) {
 result1 = dec.format(Cordx[grouping[i]]);
result2 = dec.format(Cordy[grouping[i]]);
result3 = dec.format(ClustgroupX[i]);
result4 = dec.format(ClustgroupY[i]);
 }
}
```

The main advantages of this algorithm are its simplicity and speed which allows it to run on large datasets.

XI. EXPERIMENTS

The code runs on java platform. The original number of test cases are compared with the output of the knowledge mining system. The graph is drawn. It shows the knowledge mining system reduces the size of test suite.



XII. CONCLUSION AND FUTURE WORK

In this paper we present a mining approach to have better knowledge about the test cases and about the full system. So that better test cases can be generated, selected and are used for testing. Because the main challenge in testing is to select & execute test cases. The knowledge mining of test case system has better way of mining the test suite and provides a better set of test cases to test the system performance.

In future the system can be automated by using agents.

REFERENCES

[1] Shin Yoo and Mark Harman."Pareto Efficient Multi-Objective Test Case selection", Proc. ISSTA '07, July 9-12,London, U.K. 2007 ACM.

[2] Wes Masri, Andy Podgurski. "An empirical study of test case filtering techniques based on exercising information flows", IEEE transactions on software Engineering, vol.33, No.7, July 2007.

[3] Zheng Li, Mark Harman, robert M. Hierons."Search algorithms for regression test case prioritization", IEEE transactions on software Engineering, vol.33, No.4, April 2007

[4] Dennis Jeffrey and Neelam Gupta. "Improving fault detection capability by selectively retaining test cases during test suite reduction" IEEE transactions on software Engineering, vol.33, No.2, February 2007.

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

[6] Xu, Z.; Gao, K.; Khoshgoftaar, T.M. "Application of fuzzy expert system in test case selection for system regression test", Information Reuse and Integration, Conf, 2005. IRI -2005 IEEE International Conference on volume, Issue , 15-17 Aug. 2005 Page(s): 120 – 125

[7] 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.

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

[9] Kuo –Chung Tainand Yu Lei. "A Test generation strategy for Pair-wise testing" IEEE transactions on software Engineering, vol.28, No.1, January 2002.

[10] 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.

[11] Gregg rothermal, Ronald H. Untch, Chengyun Chu, Mary Jean Harrold . "Prioritizing Test Cases for regression testing". IEEE transactions on software Engineering, vol.27, No.10, October 2001.

[12] Scott w. Ambler, IBM "Test driven development of relational databases". IEEE software May/June 2007.

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

[14] 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.

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

[16] Khaled el-Fakih, Nina Yevtushenko and gregor v. Bochmann. IEEE transactions on software Engineering, vol.30, No.7, July 2004. [17]Ryszard Janicki, Emil Sekerinski."foundations of the trac assertion method of module interface specification". IEEE transactions on software Engineering, vol.27, No.7, July 2001.