Volume 4, Issue 5, May 2014 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Intelligent Test Suite Retrieval

Bhaskar Kumar Rao. B
Department of IT,
Sree Vidyanikethan Engineering College,
Tirupati, Andhra Pradesh, India

Narendra Kumar Rao. B
Department of CSE,
Sree Vidyanikethan Engineering College,
Tirupati, Andhra Pradesh, India

Abstract— Machine Learning in software testing has been the trend of research. Currently research is progress in defect prediction, test suite minimization areas of software testing. The current paper focuses on retrieval in adaptive test suite optimization through fuzzy logic techniques. Fuzzy logic is under consideration for reason that multi constraint retrieval performs effective test case reduction.

Keywords— Test Suite, Case based reasoning, Case retrieval, Case adaptation, Fuzzy logic.

I. INTRODUCTION

Software testing is performed to identify defects in software. This is generally done in two approaches like static analysis and dynamic analysis. Static analysis is through code inspections, static analysis of code through few tools. In dynamic analysis code execution is done for identifying defects. Testing is major through dynamic techniques only. Behaviour of the system is evaluated by test cases, to verify whether system is behaving as expected. Testing is done to increase confidence of the functionality of the product or software under test (SUT).

The basic levels of testing include unit testing, integration testing, system testing and acceptance testing which help in detecting various faults. These tests are performed on the system developed based on code. Regression testing is performed on changes made to the system code under development.

The goal of testing is to carefully select test cases that have high probability of finding a defect .In black box testing requirements are the basis for selecting test cases. Black box testing may involve random selection of test cases for testing or they may take few heuristics into consideration like the following: Equivalence Class Partitioning, Boundary value analysis, Pair-wise testing, error guessing, state-based testing etc. [1].

Test Suite contains a large number of test cases to be executed on SUT.A testing framework is also known as test-harness, it provides means of defining a test suite, executing it, and reporting the results. Test Suite reduction involves minimizing the test cases which provide the coverage requirements of original suite and to remove redundant test cases.

Test suite reduction problem:

The first formal definition of test suite reduction problem introduced in 1993 by Harold et al. as follows: Given. {t1, t2,…, tm} is test suite T from m test cases and {r1, r2,…, rn} is set of test requirements that must be satisfied in order to provide desirable coverage of the program entities and each subsets {T1, T2,…, Tn} from T are related to one of ris such that each test case tj belonging to Ti satisfies ri[6,7,8,9,10,11,12,13].

Problem. Find minimal test suite T' from T which satisfies all risk covered by original suite T.

Generally the problem of finding the minimal subset T', T' is subset of T which satisfies all requirements of T, is NP-complete, because we can reduce the minimum set-cover problem to the problem of test suite minimization in polynomial time. Thus, researches use heuristic approaches to solve this problem.

Approaches in Test suite Reduction: Related work in the context of test suite reduction can be classified into two main categories: The works in which a new technique is presented and empirical studies on the previous techniques. The works proposed earlier focused includes heuristic algorithms, genetic algorithm-based techniques and approaches based on integer linear programming. In a recent study, four typical test suite reduction techniques have been evaluated and compared on 11 subject programs.

Case Based Reasoning in Test suite Optimization:

Case based reasoning solves new problems by adapting solutions to older problems and it can retain memory of previous problems, their solutions and solving new problems by reference to that knowledge. These systems generally require significantly less acquisition of knowledge, since it involves collecting a set of past experiences without the necessity of extracting a formal domain model from the previous cases.

The problem-solving life cycle in a CBR system comprises of the following four phases (4R's) [1, 2, 3]:

- Retrieve: Retrieving similar previously experienced cases whose problem is judged to be similar.
- Revise: the cases by copying or integrating the solutions from the cases retrieved.
- Reuse: adapting the solution(s) retrieved in an attempt to solve the new problem.
- Retain: new solution once it has been confirmed or validated.

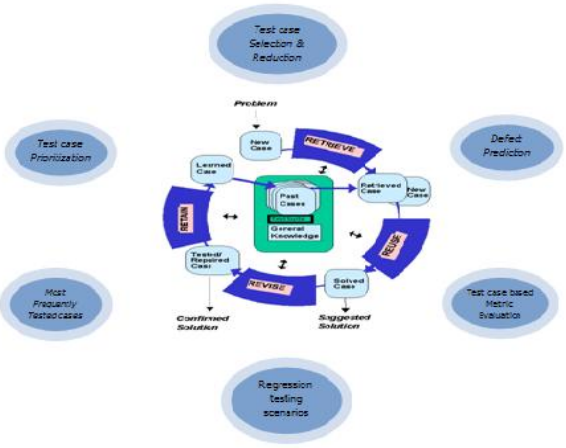

Fig. 1 Conceptual System for Adaptive Test suite with few features

As illustrated in Fig.1, Test Suite optimization technique can be fit into a CBR system where in few features which can be supported by the system are listed around the system [16, 17]. This system is capable of providing following benefits:

- Storage of test cases selected based on expertise of the tester in form of cases.
- Managing test case results for the already tested set of test cases for a module.
- Possibility of introducing the new feature of adaptation in test case selection
- Assigning priority to test cases based on criteria of testing.
- Deciding the modules to be tested and thereby suggesting test cases.
- Mapping techniques from code changes to test case selection
- Making Defect predictions in projects.
- Evaluate few metrics such as functional or test coverage metrics, software release metrics, software maturity metrics, and reliability metrics.

The main focus of this paper is on retrieval phase of the adaptive system.

Retrieval in Case based Reasoning:

Case retrieval is the process of finding out a case in a case base that are the nearer to the current case. A selection criterion is necessary to determine the best mechanism to retrieve, by determining how close the current case is to the cases stored and formulate new cases, if new cases matching to the current scenario are not available in the case base.

Cases are assumed to have two components: problem specification and solution. The problem specification is formulated out of attributes and values. The attributes of a case defines that case uniquely and should be sufficient to predict a solution for that case. The representation may be a simple flat data structure or a complex object hierarchy.

There are three general issues that are considered when creating a case base:

- The structure and representation of the cases.
- The memory model used for organizing the entire case base.
- The selection of indexes used to identify each case.

Case representation is one of the advantages in CBR, since it has support of flexible representations of cases. Following are the factors that should be considered when choosing a representation format for a case:

- Segments within the cases (i.e., internal structure) that form natural sub-cases or components. The format chosen needs to be able to represent the various forms taken by this internal structure.
- Types and structures associated with the content or features that describe a case. These types have to be available, or be capable of being created, in the case representation.
- The language or shell chosen in which to implement the CBR system. The choice of a shell may limit the formats that can be used for representation.
- The indexing and search mechanism planned. Cases have to be in a format that the case retrieval mechanism can deal with effectively.
- The form in which cases are available or obtained.

The memory model for a chosen form of case representation will depend on a number of factors as follows:

- The representation used in the case base.
- The purpose of the CBR system.
- The number and complexity of the cases being stored.
- The number of features that are used for matching cases during a search.
- Whether some cases are similar enough to group together. Where cases fall into natural groupings, some structuring facility may be useful.
- How much is known about a specific domain. This influences the ability to determine whether cases are similar.

Case indexing refers to assigning indexes to cases for future retrieval and comparison. The choice of indexes is important to enable retrieval of the right case at the right time. Suggestions for choosing indexes:

- Indexes must be predictive in a useful manner.
- Indexes should be abstract enough to allow retrieval in all the circumstances in which a case will be useful, but not too abstract.

Implementation Model of Conceptual System for test suites:

The described conceptual system can be implemented through a module diagram as proposed below. The proposed implementation is a typical scenario but it is not to be restricted to be a particular solution for the approach. The major modules in this approach are as follows:

- Decision-Classification Tree
- Case Base
- Case Based Reasoning Operations
- Adaptive System Management
- Operations and Feature support

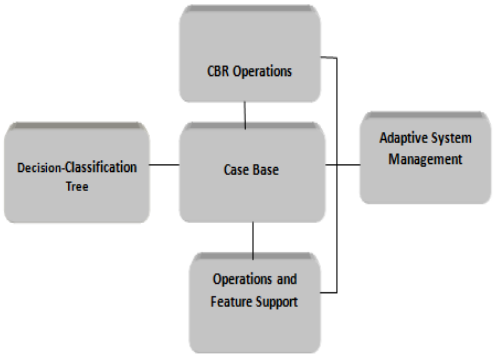

Fig-2 Module0 Diagram for Conceptual System of adaptive test suite

Decision-Classification Tree:

This structure forms the lynch-pin which holds all the components of the Conceptual System together. Basic feature of this module is storing the test cases in classified pattern in a B+ Tree. This tree classifies the test cases based on the requirements.

Requirements are classified as sub requirements (into sub trees) and its further sub components until they are not further classified. The classified requirements form the root and non-leaf structures of the B+ Tree. The leaf node of this tree comprise of the Test cases corresponding to requirements, which form the non-leaf nodes.

II. LITERATURE

Case Base:

Case base stores the stories (cases) of all the test cases selected for testing which includes the path traversed through the DC tree during the selection of test cases from the same. A typical case base comprises of attribute, value pairs which will guide through the process of selection of case along with the various test case selection tracking information like test case id, mechanism (path) followed for arriving at the test case etc... This will provide information on criteria for selection of test cases, testing methodology, reason for selection of test case, modified portion of code etc...Apart from above following are detailed benefits of storing cases:

- Storage of case scenarios of test cases used by tester.
- Storage of testers experience in case base.
- Storage of test case represents already tested set of test cases for a module.
- Deciding the mechanism for test case selection.
- Assigning priority of test cases.
- Deciding the modules to be tested.
- Making Defect predictions.

Approach for case base storage:

In case based reasoning cases are to be stored in case base. This is a crucial step in formulation of knowledge base of experience from the tester. A technique is essential for storing cases in the case base. In general case base is stored in attribute, value pairs {(u, v)}.u-represents the case scenario being tested and v-represents the corresponding test cases and path observed/followed for the retrieval of test casesbyb the tester. Depending on the detail of tester specification, test cases are retrieved from classification tree and test cases are arrived at. The test cases and requirements acquire corresponding scores during this time, which form an attribute in the node. The more the test case is retrieved the more score it is attached with, thereby indicating the increased usage of the test case.

Case Retrieval:

Case retrieval is the process of finding, within a case base, those cases that are the closest to the current case. To carry out effective case retrieval, there must be selection criteria that determine how a case is judged to be appropriate for retrieval and a mechanism to control how the case base is searched. The selection criteria are necessary to determine which is the best case to retrieve, by determining how close the current case is to the cases stored. The case selection criteria depend partly on what the case retriever is searching for in the case base. Most often the case retriever is searching for an entire case, the features of which are compared to those of the current case. However, there are times when only a portion of a case is being sought. This situation may arise because no full case exists and a solution is being synthesized by selecting portions of a number of cases. A similar situation is when a retrieved case is being modified by adopting a portion of another case in the case base.

Retrieval is a major research area in CBR. The most commonly investigated retrieval techniques, by far, are the k-nearest neighbours (k-NN), decision trees, and their derivatives. These techniques involve developing a similarity metric that allows closeness (i.e., similarity) among cases to be measured.

Nearest-neighbour retrieval:

In nearest-neighbour retrieval, the case retrieved is chosen when the weighted sum of its features that match the current case is greater than other cases in the case base. In simple terms with all features weighted equally, a case that matches the present case on n features will be retrieved rather than a case that matches on only k features, where $k < n$. Features that are considered more important in a problem-solving situation may have their importance denoted by weighting them more heavily in the case-matching process.

Inductive approaches

When inductive approaches are used to determine the case-base structure, which determines the relative importance of features for discriminating among similar cases, the resulting hierarchical structure of the case base provides a reduced search space for the case retriever. This may, in turn, reduce the query search time.

Knowledge-guided approaches:

Knowledge-guided approaches to retrieval use domain knowledge to determine the features of a case that are important for retrieving that case in the future. In some situations, different features of a case will have different levels of importance or contribution to the success levels associated with that case. As with inductive approaches to retrieval, knowledge-guided indexing may result in a hierarchical structure, which can be more effective for searching.

Validated retrieval

There have been numerous attempts at improving retrieval. One of these is validated retrieval, which consists of two phases. Phase 1 involves the retrieval of all cases that appear to be relevant to a problem, based on the main features of the present case. Phase 2.Involves deriving more discriminating features from the initial group of retrieved cases to determine whether these cases are valid in the current situation. The advantage of validated retrieval is that inexpensive computational methods can be used to make the initial retrieval from the case base, while more expensive computational methods can be used in the second phase, where they are applied to only a subset of the case base.

Factors to consider when determining the method of retrieval:

- The number of cases to be searched
- The amount of domain knowledge available.
- The ease of determining weightings for individual features.
- Whether all cases should be indexed by the same features or whether each case may have features that vary in importance.

Once a case has been retrieved, there is usually an analysis to determine whether that case is close enough to the problem case or whether the search parameters need to be modified and the search conducted again. If the right choice is made during this analysis, there can be a significant time saving. For example, the adaptation time required for a distant case could be significantly greater than searching again. When considering an analysis method for this decision, the following points should be considered:

- The time and resources required for adaptation
- The number of cases in the case base (i.e., how likely it is that there is a closer case)
- The time and resources required for search
- How much of the case base has already been searched.

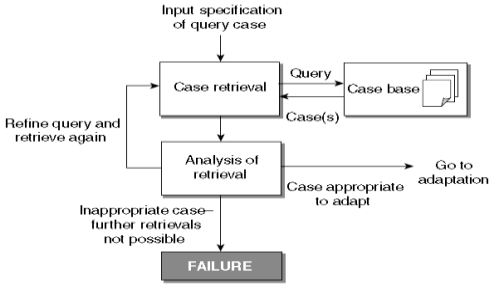

Fig-2 Case Retrieval

Case Adaptation

Case adaptation is the process of transforming a solution retrieved into a solution appropriate for the current problem. It has been argued that adaptation may be the most important step of CBR since it adds intelligence to what would otherwise be simple pattern matchers.

A number of approaches can be taken to carry out case adaptation:

- The solution returned (case retrieved) could be used as a solution to the current problem without modification, or with modifications where the solution is not entirely appropriate for the current situation.
- The steps or processes that were followed to obtain the earlier solution could be rerun without modification or with modifications where the steps taken in the previous solution are not fully satisfactory in the current situation.
- Where more than one case has been retrieved, a solution could be derived from multiple cases or, alternatively, several alternative solutions could be presented.

Adaptation can use various techniques, including rules or further case-based reasoning on the finer-grained aspects of the case. When choosing a strategy for case adaptation, it may be helpful to consider the following:

- On average, how close will the case retrieved be to the present case?
- Generally, how many characteristics will differ between the cases?
- Are there common sense or otherwise known rules that can be used when carrying out the adaptation?

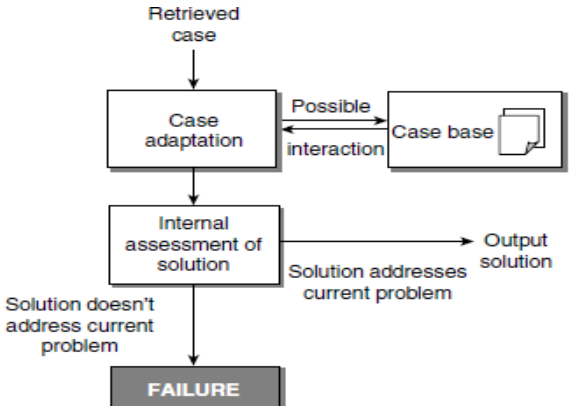

Fig-3 Case Adaptation

Adaptation Methods

Here we briefly explain the adaptation methods found in the survey [4, 5].

Adaptation using genetic algorithm

In a genetic algorithm the case bases form the initial population of genotypes. The algorithm first retrieves partial matching cases from case base with specified design requirements. Next the retrieved cases are mapped into a genotype representation. Then crossover and mutation operators are applied. Finally, newly generated genotypes are mapped into corresponding phenotypes/cases by inferring values for the attributes and adding the context of the new design.

Adaptation with Bayesian networks

In this approach, the knowledge of the problem domain is coded using m attributes A1, . . . ,Am, where an attribute Ai has ni possible values, ai1, . . , ain . As a result the world consist of binary vectors in which the characteristic function fi ~aij ! is 1 if Ai has value aij ; otherwise fi aij ! is 0. A case Ck is coded as a vector Pk[a][m], . . . ,Pk[a][m] , where P[k]a[i][j] expresses the likelihood for attribute Ai to have value a[i][j] in class k The case base consists of l cases say C1, . . . ,Cl , each of which is provided with a unique label ck.

Adaptation guided by constraint satisfaction

This is a general method for solving case adaptation problems for the large class of problems that can be formulated as constraint satisfaction problems i.e. CSP. It consists of a number of choices that need to be made (variables,) each of which has an associated number of options the variable domain and a set of relationships between choices constraints. A valid solution to a CSP is an assignment of a value to each variable from its domain with the total set of assignments respecting all the problem constraints. The adaptation method is based on the concept of interchange ability between values in problem solutions. The method is able to determine how change propagates in a solution set and generate a minimal set of choices, which need to be changed to adapt an existing solution to a new problem.

Substitution-based adaptation model

In this method a domain-independent model is presented to structure the knowledge, where adaptation knowledge is explicitly represented. Built upon this model, description logics. Inference mechanism is used to formalize an adaptation scheme based on substitutions, in which the search for substitutes is guided by a set of memory instructions. The system includes a learning component that records the successful search episodes as a search heuristic. The search heuristics include slots to store instances of origin, destination, and the path. To record the number of times a heuristic has been successfully applied, the weight slot is used. Thus when searching for substitutes, the applicable heuristic with the highest level is selected.

Derivational replay

In this adaptation method, old solutions are used to fit a new situation. In Ref. 7, adaptation cases and memory search cases are used to store the information about the adaptation problem as a whole and the way it was solved. Learning algorithms are used to learn knowledge from the previous cases.

Substitutional adaptation

This is the process in which some parts are replaced. Although the previously explained adaptation is also based on substitution, it uses memory organization and is domain independent. Substitution methods can be of six types:

I) Reinstantiation
II) Parameter adjustment
III) Local search
IV) Query memory
V) Specialized search
VI) Case-based substitution

Fuzzy logic implementation for retrieval in Cases

A fuzzy logic system (FLS) can be defined as the nonlinear mapping of an input data set to a scalar output data. A FLS consists of four main parts: fuzzifier, rules, inference engine, and defuzzifier. These components and the general architecture of a FLS are shown in following

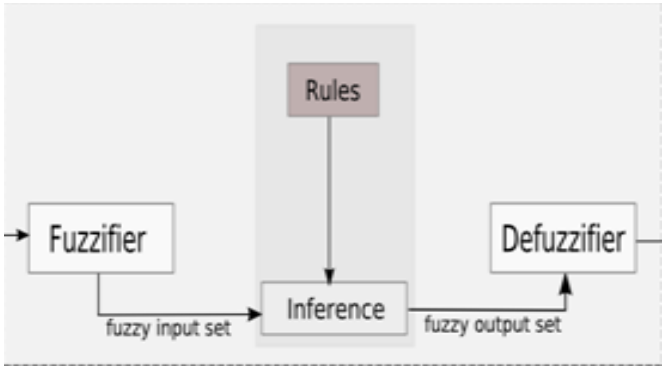

Fig.4 A Fuzzy Logic System

III. PROPOSED SYSTEM

Problem Statement

- During Software Testing ,Tester faces a daunting task of selecting effective test cases to perform situation intensity based testing ,for which tester needs to select test cases after going through every test case in the test suite.
- In course of time the likely chance of missing the right test cases or important test cases are high.
- An Automated adaptive approach for test suite optimization is very much required in this case, to remember the past experience, learn and be able to suggest the same to the tester, to suggest effective and situation intensity based reduced test cases [15].

Objectives

The Main objectives of the Retrieval in Adaptive Test suite are shown below:

- Providing the possible best test case in minimum amount of time.
- Test Suite Optimization in Selection of effective Test cases for testing, while keeping track of adequacy criteria.
- A Case Based Reasoning (CBR) approach is selected for the previous specified case to suggest test cases adapting to the test scenarios during testing phase especially during system testing with less domain Knowledge.

Scope

The scope of the Retrieval in Adaptive Test suite is Optimization of Test suite in the process of retrieval of test cases for testing the software program, finds it scope in all the Software Engineering projects using the Case Based Reasoning approach.

Applications

The applications of the Retrieval in Adaptive Test suite are:

- Retrieve the Test cases from case automatically.
- Suggest the Tester while selecting the test cases.
- Proposed system stores the previous test case and will give in future if any need.

Implementation Details:

Implementation is the stage of the project when the theoretical design is turned out into a working system. Thus it can be considered to be the most critical stage in achieving a successful new system and in giving the user, confidence that the new system will work and be effective.

The implementation stage involves careful planning, investigation of the existing system and it's constraints on implementation, designing of methods to achieve changeover and evaluation of changeover methods.

Modules:

The system is proposed to have the following modules.

1. Exporting test suite into the database.
2. Generating B+ Tree
3. Generation of cases
4. Retrieval of Cases
5. Adaptation of cases

Exporting test suite into the database

Choosing the CSV file from the system which containing the requirements and test case with the associated weights. The contents are exported into the database with a specific filename.

Generating B+ Tree

A B+ tree is an n-array tree with a variable but often large number of children per node. A B+ tree of order v consists of a root, internal nodes and leaves. The root may be either leaf or node with two or more children.

A B+ tree can be viewed as a B-tree in which each node contains only keys (not pairs), and to which an additional level is added at the bottom with linked leaves.

The primary value of a B+ tree is in storing data for efficient retrieval in a block-oriented storage context in particular, file systems. This is primarily because unlike binary search trees, B+ trees have very high fan-out (typically on the order of 100 or more), which reduces the number of I/O operations required to find an element in the tree.

Search

The root of a B+ Tree represents the whole range of values in the tree, where every internal node a subinterval. We are looking for a value k in the B+ Tree. Starting from the root, we are looking for the leaf which may contain the value k. At each node, we figure out which internal pointer we should follow. An internal B+ Tree node has at most $d \leq b$ children, where every one of them represents a different sub-interval. We select the corresponding node by searching on the key values of the node.

This pseudo code assumes that no duplicates are allowed. An easy way to comply with the conference paper formatting requirements is to use this document as a template and simply type your text into it.

Generation of cases

The software tester searches the test cases by specifying the requirements. After getting all the test cases he may delete or add furthermore test case if he want. After completing all the process he saves it into the database as a new case along with his name and automatically generated case ID.

Retrieval of Cases

The tester specifies the query by means of requirements. The requirements are taken into consideration while retrieving all the related matched cases for the query he specified. This phase is also called analysis.

The retrieval happens by using Fuzzy logic technique.

Adaptation of cases

After retrieving all the matched cases, the best cases are adopted to propose the solution for the current new problem, if the exact match is found then it will be proposed.

The Role of fuzzy logic

Firstly, a crisp set of input data are gathered and converted to a fuzzy set using fuzzy linguistic variables, fuzzy linguistic terms and membership functions. This step is known as fuzzification.

Afterwards, an inference is made based on a set of rules. Lastly, the resulting fuzzy output is mapped to a crisp output using the membership functions, in the defuzzification step.

Algorithm for Fuzzy logic algorithm:

1. Define the linguistic variables and terms (initialization)
2. Construct the membership functions (initialization)
3. Construct the rule base (initialization)
4. Convert crisp input data to fuzzy values using the membership functions (fuzzification)
5. Evaluate the rules in the rule base (inference)
6. Combine the results of each rule (inference)
7. Convert the output data to non-fuzzy values (defuzzification).

Membership Functions:

A membership function describes the degree of membership of a value in a fuzzy set.

Fuzzy logic in our Current Work

Fuzzification

- Retrieve the matched cases from the case base.
- Convert the case weight numerical value into the crisp value
- This phase generates a fuzzy input set.
- Build the Fuzzy Rules (Inference)
- Assign zero value to the unused requirements in the retrieved case with respect to given input requirements.
- Adjust the given input to the value 100 % by reducing the requirement value with equal priority.

- Generating the fuzzy input requirement value [0,1]

Defuzzification

- Fuzzy output set in given as input to this phase. i.e., the best case from retrieved cases. Converting the fuzzy output to the Boolean value for the acceptance of cases.
  1. Input: Given the input requirements as **{r1, r2,}**
  2. Retrieve all the matched cases from the case base with respect to the given input.
  3. Finding out the value for each matched case with respect to the given input requirements
  4. Example: matched Case = **{r1, r2, r3…}**
  5. Assigning zero value to the unnecessary requirement
     {r1, r2, 0 ...} (rule1)
  6. Identifying the r1, r2 … values in each matched retrieved case
  7. Adjust the given input to the value 100 % by reducing the requirement value with equal priority.
  8. Adding all case requirement values to generate a case value between 0.0 to 1.0 ranges
  9. Member function = $$\mu(x) = \begin{cases} 1, & l \leq x \leq r \\ 0, & \text{otherwise} \end{cases}$$
  10. Retrieving all the matched case between the given ranges low to high.

Sample Scenario:

Input= {r1, r2}
Retrieved cases = {(c45) (c51) (c68) (c71)}
Identifying case value for each case
Example: c45 (r1, r2, r3, r4)
According rule1 assigning (r3, r4) =0
According rule2 calculating the c45 value using r1=0.9, r2=0.6 values adjusting them to 100%
Now c45 = (r1=0.45, r2=0.3, r3=0, r4=0)
According to rule3 adding all value to get c45 value
c45 = 0.45+0.30+0.0+0.0 = 0.75
Similarly identifying all the remaining case values as
c51=0.65
c68= 0.35
c71=0.95
Now giving the range to filter cases
Range= 0.50-0.80
Retrieved cases = {c45, c51}

Selecting test cases

Once the case have been chosen, the problem reduces to selecting test cases from the clusters. This can be performed by ranking of requirements from the selected cases and selecting all corresponding requirements from the cases to a value specified in the range requested[14].

IV. EXPERIMENT AND ANALYSIS

Subject Application & Metrics

SIR repository based *space* program was used in this work as program data. It has a test pool of nearly 1000 test cases and 38 versions of the same program containing faults.

Current work compares the effectiveness of proposed work with random reduced test suites for its evaluation. Following are the metrics to be observed:

Percentage of size reduction and percentage fault detection reduction has been calculated over the entire experiment to measure the effectiveness of the proposed method over random experiments.

$$TS_{(\%\ Red)} = 1 - \frac{|T_{rs}|}{|T|} \quad X \quad 100$$

|T|- Total number of test cases in the original suite and $|T_{rs}|$ represents test cases in representative set.

$$FD_{(\%\ Red)} = 1 - \frac{|FD_{Reduced}|}{|FD_{Full}|} \quad X \quad 100$$

Percentage reduction in fault detection $FD_{(\%\ Red),}$ $FD_{Reduced.Defect}$ detection from reduced suite, FD_{Full} is complete suite.

Method of Experimentation

The defects were detected from the suite by selecting a given number of test cases by applying the test case selection approach and detect the fault detection effectiveness from Random and CBR retrieval.

Table 1. Size vs Faults

Means over Test Suites					
Original Suite		**Random reduced**		**CBR Retrieval**	
Size	**Faults Detected**	**Size**	**Faults Detected**	**Size**	**Faults Detected**
250	25	60	18	52	**20**
% Reduction from Original		**76**	**28**	**79.2**	**20**

Form the results it is observed that our approach is slightly better than the random approach of test case selection for testing and fault detection and size reduction have been slightly encouraging. This also encourages towards fact that few other more effective case retrieving methods can improve the performance. The graphs representing (Fig-5) the above scenario in terms of fault detection effectiveness is as follows:

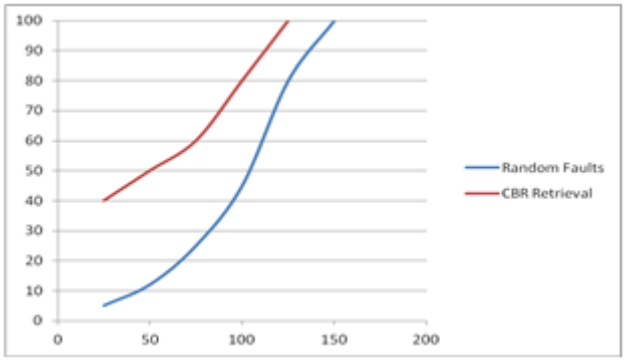

Fig-5. Performance in fault detection

Benefits:

From the graph it can concluded that faults were detected early compared to random suites and early detection leads to early fixing of issues and a cost effective effort.

Performance Evaluation of Results

Retrieval in adaptive test suites provides high performance than Existing system. The system provides optimal test suite with high speed and accuracy.

The software provides high security of data because we use DB2 database to save the data. The security is provided by password giving to the tester. Compare with the existing system Performance of the system is increased. As much time of retrieval is less, as such performance has considerably increased in the proposed system.

V. CONCLUSIONS

In our project, we generated a test suite in the form of a B+ tree so that all leaf node i.e., test cases are at the same level. Tester can retrieve the matched cases from the database. If the exact case is not found the best among the matched case is adapted to the current problem, further we can integrated the new case into the database as anew case. In this project we developed only the First Phase of CBR i.e. Retrieve. This considerably reduces the size of test suite by selecting the cases from the case.

REFERENCES

[1]. RAMON LOPEZ DE MANTARAS" Retrieval, reuse, revision and retention in case-based reasoning" The Knowledge Engineering Review, Vol. 20:3, 215–240, 2006.
[2]. Selma Limam Mansar "Case-Based Reasoning as a Technique for Knowledge Management in Business Process Redesign", Academic Conferences Limited, 2003.
[3]. Padraig Cunningham "Case-Based Reasoning in Scheduling: Reusing Solution Components" International Journal for Production Research, 1997.
[4]. LORIA "Application of the Revision Theory to Adaptation in Case-Based Reasoning: the Conservative Adaptation", ICCBR, 2007.
[5]. A. Aamodt, E. Plaza "Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches" AI Communications. IOS Press, Vol. 7: 1, pp. 39-59, 1994.
[6]. Dennis Jeffrey "Test Suite Reduction with Selective Redundancy", ICSM'05.
[7]. James A. Jones "Test-Suite Reduction and Prioritization for Modified Condition/Decision Coverage"IEEE transactions, March 2003.
[8]. Mats P.E. Heimdahl "Test-Suite Reduction for Model Based Tests: Effects on Test Quality and Implications for Testing", Proceedings 19th International Conference on Automated Software Engineering, 2004.
[9]. Kartheek Muthyala "A Novel Approach to Test Suite Reduction Using Data Mining", IJCSE, vol-2, June 2011.

[10]. Lilly Ramesh "An Efficient Reduction Method for Test Cases", IJEST, vol-2, 2010.
[11]. Hadi Hemmati "Empirical Investigation of the Effects of Test Suite Properties on Similarity-Based Test Case Selection", ICST '11
[12]. G. Rothermel, M.J. Harrold, J. von Ronne, C. Hong, "Empirical Studies of Test-Suite Reduction", Journal of Software Testing, Verification, and Reliability, 12(4), 2002, pp. 219-249. On Software Maintenance, IEEE Computer Society, 1998.
[13]. G.Rothermel, M.J. Harrold, J. Ostrin, C. Hong, "An Empirical Study of the Effects of Minimization on the Fault Detection Capabilities of Test Suites", Proceedings of the International conference .
[14]. M.J. Harold, R. Gupta, M.L. Soffa, "A Methodology for Controlling the Size of a Test Suite", ACM Transactions on Software Engineering Methodologies 2, 1993, pp. 270-285.
[15]. T.Y. Chen, M. F. Lau, "Heuristics toward the Optimization of the Size of a Test Suite" Proc. 3rd Int'l Conf. on Software. Quality Management. Vol. 2, Seville, Spain, April 1995, pp. 415-424.

