Features of Adaptive Test Suites

Narendra Kumar Rao B. and RamaMohan Reddy A.

[1]*Dept. of CSE, Sree Vidyanikethan Engineering College, Tirupati, India*
[2]*CSE, Sri Venkateswara University, Tirupati, India*

Abstract

Case-based reasoning is an approach to problem solving and learning that has got a lot of attention over the last few years. This paper provides an overview of the case-based reasoning, brief outline on applicability of case-based reasoning to test suite, describing the various features of adaptive test suite system based on CBR framework will possess, focuses on benefits of adaptive approach over existing test case selection/retrieval approaches. The current work demonstrates the benefit of case base in reasoning approach for selecting test case or for testing the product during system testing and regression testing phases of the product.

Keywords: *Retrieve, Reuse, Revise, Retain, Test suite reduction, Test case Prioritization, Test case Selection, Test suite Optimization*

1. Introduction

In Case Based Reasoning, problems are solved by adapting the solutions of similar previous cases stored in a case memory. CBR mainly consist of four phases they are Retrieve, Reuse, Revise, and Retain [1]. Retrieving the cases from the case base whose problem is most similar to the new problem. Reusing is the solutions from the retrieved cases to create a proposed solution for the new problem. Revising the proposed solution to take into account of the differences between the existing problems, current problems and modify the same in accordance with the current problem. Retaining the new problem and its revised solution as new case for new case for case-base if appropriate. CBR is has few assumptions, first and main assumption is that "Similar problems have similar solutions". And other assumptions are "The world is a regular place"- what holds true today will probably hold true tomorrow. Next is "Situation repeat"- if they do not there is no point in remembering them.

Test suite is a collection of test cases that intended to be used to test a software program to show that it has some specific set of behaviors. A Test case is representation of test scenarios. 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. The main contribution of the work is Classification of techniques for R4 phases in CBR, Test suite optimization, Case Based reasoning and Test suite optimization.

Markovitch & Scott (1993) propose a unifying framework for the systematic discussion of all of the various strategies for coping with harmful knowledge in general, and the utility problem in particular. Their framework is based on different types of filters for eliminating harmful knowledge at various stages in the problem solving cycle. One approach that is especially relevant in CBR is to simply delete harmful cases from the case base so that they cannot actively contribute to ongoing problem solving costs deletion policies in CBR correspond to selective retention filters in the Markovitch & Scott framework. Surprisingly enough, in many speed-up learners even

the apparently naive random deletion of knowledge items (to maintain the knowledge base to some predefined size) works quite well for optimizing efficiency. Even though random deletion removes both useful and redundant items it can equal the success of more sophisticated methods.

2. Overview of Case-Based Reasoning

The processes involved in CBR can be represented by (Figure 1). Aamodt and Plaza have described CBR typically as a cyclical process comprising *the four Rs*:

- RETRIEVE the most similar case(s);
- REUSE the case(s) to attempt to solve the Problem
- REVISE the proposed solution if necessary, and
- RETAIN the new solution as a part of a new case.

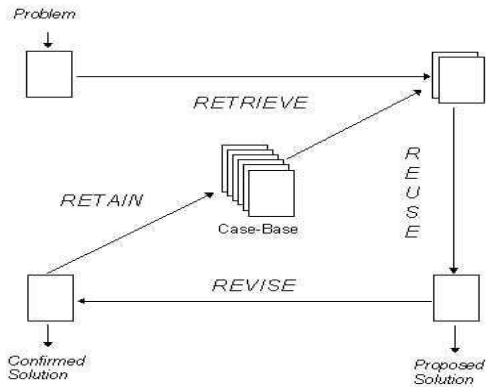

Figure 1. The CBR Cycle

A new problem is matched against cases in the case base and one or more similar cases are retrieved. A solution suggested by the matching cases is then reused and tested for success. Unless the retrieved case is a close match the solution will probably have to be revised producing a new case that will be retained.

Few Important terminologies related to CBR are as follows:

Case Representation:

A case is a contextualized piece of knowledge representing a particular experience. It contains the past lesson that is the content of the case and the context in which the lesson is to be used. Typically a case comprises of following:

- problem that describes the state of the world when the a case occurs,
- solution which states the derived solution to that problem, and/or
- outcome which describe the state of the world after the case has taken place.

Indexing

Case indexing involves assigning indices to cases to facilitate case retrieval. Indices should follow few properties:

- predictive,
- address the purposes the case will be used for,
- allow for widening the future use of the case-base, and
- concrete enough to be used in future.

Both manual and automated methods have been used to select indices. Choosing indices manually involves deciding a case's purpose with respect to the aims of the reasoner and deciding under what circumstances the case will be useful. Despite the success of many automated methods, it is believed that people tend to do better at choosing indices than algorithms.

Induction

Induction algorithms (e.g. ID3) determine which features do the best job in discriminating cases, and generate a decision tree type structure to organize the cases in memory. This approach is useful when a single case feature is required as a solution, and where that case feature is dependent upon others.

Adaptation

Once a matching case is retrieved a CBR system should adapt the solution stored in the retrieved case to the needs of the current case. Adaptation looks for prominent differences between the retrieved case and the current case and then applies formulae or rules that take those differences into account when suggesting a solution. In general, there are two kinds of adaptation in CBR:

Structural adaptation, in which adaptation rules are applied directly to the solution stored in cases. This kind of adaptation is used in JUDGE and CHEF.

Derivational adaptation that reuses the algorithms, methods or rules that generated the original solution to produce a new solution to the current problem. In this method the planning sequence that constructed that original solution must be stored in memory along with the solution as in MEDIATOR.

An efficient CBR system may need both structural adaptation rules to adapt poorly understood solutions and derivational mechanisms to adapt solutions of cases that are well understood.

Few case based reasoning adaptation techniques are as follows:

- Null adaptation
- Parameter adjustment
- Abstraction and respecialisation
- Critic-based adaptation
- Re-instantiation
- Derivational replay

- Model-guided repair

TEST SUITE:

In software development, a test suite, less commonly known as a validation 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 behaviors. 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.

Test suite reduction problem:

The first formal definition of test suite reduction problem introduced in 1993 by Harrold 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.

*Problem***.** Find minimal test suite *T'* from *T* which satisfies all *ri*s 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.

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 which propose a new approach commonly include 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. Based on the results, this study suggests that the heuristic H must be the first choice when selecting from the reduction techniques. Testing criteria are defined in order to help the selection of subsets of the input domain to be covered during testing. For example, a code coverage criterion provides test suite adequacy with respect to coverage of the program entities and also provides a check on its quality. Assuming testing criterion *C* which satisfies by the test suite *T*, a test case, *t*, is *redundant* if the suite *T*-{*t*} also satisfies *C*. Therefore, removing those test cases which are redundant with respect to some specific criteria preserves test suite's adequacy with respect to that criteria. In previous empirical studies researchers commonly applied various code coverage criteria in their reduction techniques. The results of empirical studies show that the more percentage of the test suite size is reduced the more percentage of faults will be lost. Usually a test suite reduction technique attempts to remove *redundancy* among test cases and retain the most *effective* ones into the test suite. Effective test cases are those that are capable of satisfying the most requirements as well as exposing the most of the existing faults. Note that the more requirements are satisfied, the more execution paths within the program would be exercised which yields the all kinds of the faults to be exposed.

TEST CASE CLASSIFICATION:

System test cases cover the application from end to end. It is not based on the requirements given by the client. We need to cover the whole application. Test cases were classified into different types:

Graphical user interface testing
Usability testing
Performance testing
Compatibility testing
Error handling testing
Load testing
Volume testing
Stress testing
Security testing
Scalability testing
Sanity testing
Smoke testing
Exploratory testing
Ad hoc testing
Regression testing
Reliability testing
Installation testing
Maintenance testing
Recovery testing and failover testing.

3. Applicability of Case-based Reasoning to Test Suite

The Case Based Reasoning as such can be tailored to meet the necessary condition for test suite reduction. The four Phases involved can be designed as below:

Retrieve: The retrieval case can only proceed once there is a proper structure storing the desired case in required format suitable for retrieval. In this direction a tree based approach is designed for the case. A partial DC tree is given in Fig-2, a Decision and Classification tree is constructed. A DC tree comprises of system test cases arranged based system artifacts of the following (typical but not restricted to):

- Type of system testing
- Modules of the system
- Dependency between modules
- Interfaces involved
- Classes involved
- Dependency between classes.

In Case based reasoning the cases are constructed based on the above defined artifacts as attributes. The system test cases obtained from requirements are arranged as the separate case in low order of the tree. In fact a tree is constructed based on the above attribute and values. A system test case can fall under a particular system testing, module of a system, dependency between modules, interface, classes involved, and dependency among classes and so forth which can be sub case for the major case.

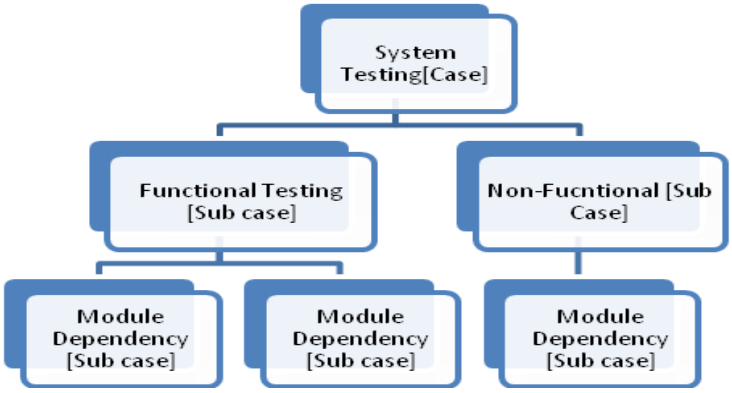

Figure 2. Partial DC Tree

Specifically the test cases can be decided by the tester of the system test cases when he is performing it manually. Anyways this CBR approach will help in identifying the test cases based on the specified details like the following. A particular objective of system testing may be selected by the tester of system test cases and this will list out all the system test cases required for fulfilling the particular objective. On further clarity/querying in information provided by the tester like the modules of the system to be tested, Dependency between modules, interface, classes involved, dependency among classes and so forth the retrieve can result in identifying a proper test case of choice which the tester may be looking for. In detail, further approach can be extended to index system test cases based on a system test case Identifier, which comprise of all the attributes forming the case. In this case the identification can be very easy for classification. Also any new case can be added further. Figure 3 signifies the same.

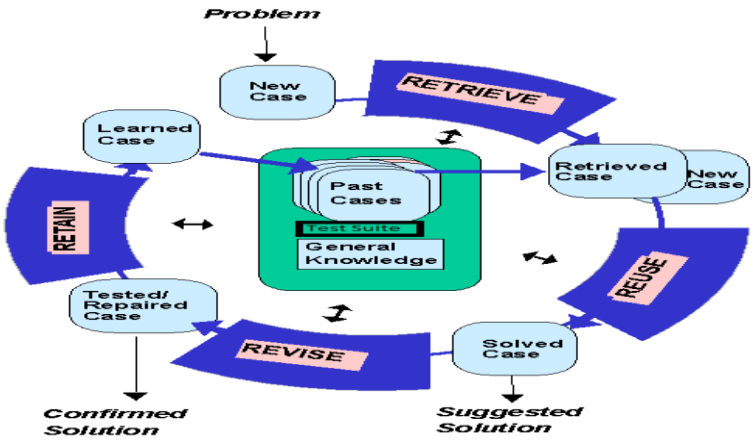

Figure 3. Phases of the Test Suite Optimization based on Case Based Reasoning

On further clarity/querying in information provided by the tester like the modules of the system to be tested, Dependency between modules, interface, classes involved, dependency among classes and so forth the retrieve can result in identifying a proper test case of choice which the tester may be looking for. In detail, further approach can

be extended to index system test cases based on a system test case Identifier, which comprise of all the attributes forming the case. In this case the identification can be very easy for classification. Also any new case can be added further. Fig-3 signifies the same.

Reuse The retrieved set of test case forms a basis for the current scenario can also be taken up as a new case or sub case as defined previously. Any set of system test cases that are previously tested can also be used during regression testing. The result of system test can be any one of the following passed, failed, not yet tested. The system test cases which were previously distinguished as a test case can be reused again, thereby reusing test cases.

Revise The revision involves the testing of system test cases manually or automatic testing and thereby changing the status to one of the following states ie. Tested or not tested again tested can have two more sub states like tested but passed, tested but failed. The test cases can be revised among them so that not only test case states can be changed but the cases can be added, duplicate system test cases can be eliminated, new cases can be added and so on.

Retain The test cases which were revised earlier can now form a part of newly created case/sub case or a old case/sub case being created. This can further be updated in the DC tree mentioned in earlier stage of DC tree.

4. Features of Adaptive Test Suite System based on CBR Framework

The Intended adaptive test suite system acts as a repository where System test cases are arranged in form of DC tree. The designed system along with DC tree is accessible to Tester(s).Tester who wants to test a system under development has to specify details regarding the following:

- Type of system testing
- Modules of the system
- Dependency between modules
- Interfaces involved
- Classes involved
- Dependency between classes, etc

Based on more and more details a tester specifies out of the above (because test cases are arranged in above format only in the DC tree).The system is capable of doing the following:

Test case Selection and Reduction

The adaptive system can guide a tester in selection of test cases, when he is specifying the type of testing type of system testing, modules of the system, dependency between modules, interfaces involved, classes involved, dependency between classes, etc..The selected test cases are reduced by complexity of number of branches of tree at any given level in DC tree.

Store the case history of Tester(s).

- A Tester arriving at the untrained system can select all the available test cases that the CBR system suggests, later on the tester starts specifying the type of testing type of system testing, modules of the system, dependency between modules, interfaces involved, classes involved, dependency between classes, etc..
- More details the tester specifies he narrows down on the number of test cases in hand for the tester for selection.
- The entire history of the test case selection is stored in case base under a specified case name specified by Tester (name being a non-significant identifier).
- On further testing, scenarios which are not relevant to existing cases in case base, new cases are stored in case base.
- If a relevant scenario of testing happens to be repeated by any current tester, which is already in case base, the adaptive system comes into being suggesting few other matching cases by previous testers. Now the current tester can opt to select a new set of system test cases or a previously used set of case, which is suggested by the adaptive system.

Most Frequently Tested cases/Scenarios/Modules/bugs

Based on number of times a test case a particular stored test case in case base is being accessed, frequency can be assigned to test cases as such the help in understanding the importance of test case for testing of module/interface. Hence frequency of execution of case(of case-base)/test case/Scenario/module/bug verification is notified based on frequency count being assigned to test case ,whenever it is accessed/deemed more frequently by the tester.

Test case Prioritization

Test case prioritization techniques schedule test cases for execution in an order that attempts to increase their effectiveness to meeting some performance goal. Such a case-wide sequence of test case can be notified from weightage methods, sequencing method etc.which are few of prioritization techniques that are supported. Also they can represent severity of defects/test cases. Such test cases can really help during sanity/release testing of modules to the client. Lesser critical tester (resource) involvement is required even critical release of product to client, which may reduce project costs. Any of familiar priority based technique may be used for this case.

Test case based Metric Evaluation

Following are few test case metrics that can be evaluated using the above specified module in adaptive test case Conceptual System:

- Functional or Test Coverage Metrics.
- Software Release Metrics.
- Software Maturity Metrics.
- Reliability Metrics.

These metrics are helpful in deciding the type of testing required, type of bugs found during test, severity of bugs, stability of module etc.

Defect Prediction

The test cases and the corresponding results help in identifying the defects in classes, modules, interfaces, system testing etc..A Naïve bayes' or equivalent approach may suggest or help in concluding the defect prediction.

Regression testing scenarios

The basic feature of adaptively to test cases is effective during regression testing, because they help in identifying the particular type of testing/test case required to test the module/class etc..A tester can get significant help during regression testing while it suggests about results of test cases previously tested, any modular changes affecting the test case results may easily be notified to the tester, which is significant help to tester and improves productivity. A detailed discussion can correlate to a separate project on itself.

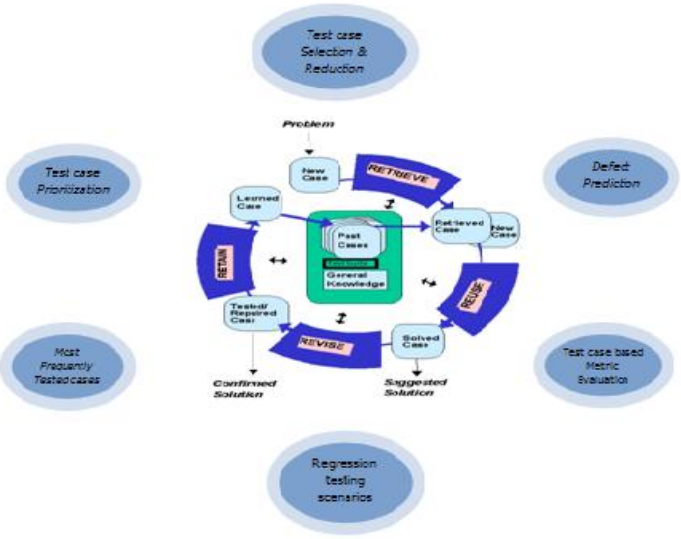

Figure 4. Conceptual System for Adaptive Test Suite with Few Features

Above specified features form a Conceptual System for adaptive test suite optimization.

i. Benefit of case base in reasoning approach for selecting test cases

Case based reasoning approach is based on the fact on how human beings think. Case base in adaptive test suite has the following features:

- Storage of test cases used by tester.
- Storage of testers experience in form of case in case base.
- Storage of test case results for the 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 in projects.

5. Conclusion and Future Work

In this paper, we described about the Case Base Reasoning and the phases involved in it, features that can be used with adaptive test suite optimization, benefits of test case base etc. Further with respect to any decision tree based System test case selection algorithm can be used. The same adaptive technique can further be used to suggest the type of test case selection algorithm to be used for a particular type of system test case selection. A detailed discussion on this can further formulate a approach of case based reasoning applied to choosing a test case selection, prioritization method selection for a particular domain of product under development viz.. embedded, Web based, SAP based, Cloud based etc.. Each feature specified in attachment with adaptive test suites can further promulgate to individual research items in themselves for various domains of Projects under development. Regression is vast feature out of the above that can be exploited to maximum extent.

A Work of this nature is essential in wake of tools present for generation, execution of tools already available, but this work focuses on transfer of human knowledge, expertise, testing based on results of previous version of test cases executed, effective resource utilization(tester) in wake of reduced and effective execution of reduced set of test cases. This will further reduce testing costs to a greater extent in combination with other approaches already suggested by many other researchers.

References

[1] P. Cunningham, "Case-Based Reasoning in Scheduling: Reusing Solution Components", International Journal for Production Research, **(1997)**.
[2] J. A. Jones, "Test-Suite Reduction and Prioritization for Modified Condition/Decision Coverage", IEEE transactions, **(2003)** March.
[3] K. Muthyala, "A Novel Approach To Test Suite Reduction Using Data Mining", IJCSE, vol. 2, **(2011)** June.
[4] L. Raamesh, "An Efficient Reduction Method for Test Cases", IJEST, vol. 2, **(2010)**.
[5] M. J. Harrold, R. Gupta and M. L. Soffa, "A Methodology for Controlling the Size of a Test Suite", ACM Transactions on Software Engineering Methodologies, vol. 2, Angeles, **(2003)**, pp. 270-285.
[6] S. L. Mansar, "Case-Based Reasoning as a Technique for Knowledge Management in Business Process Redesign", Academic Conferences Limited, **(2003)**.
[7] LORIA, "Application of the Revision Theory to Adaptation in Case-Based Reasoning: the Conservative Adaptation", ICCBR, **(2007)**.
[8] A. Aamodt and E. Plaza, "Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches", AI Communications, IOS Press, vol. 7, no. 1, **(1994)**, pp. 39-59.
[9] D. Jeffrey, "Test Suite Reduction with Selective Redundancy", ICSM'05, **(2005)**.
[10] M. 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)**.
[11] H. Hemmati, "Empirical Investigation of the Effects of Test Suite Properties on Similarity-Based Test Case Selection", ICST '11, **(2011)**.
[13] G. Rothermel, M. J. Harrold, J. von Ronne and C. Hong, "Empirical Studies of Test-Suite Reduction", Journal of Software Testing, Verification, and Reliability, vol. 12, no. 4, **(2002)**, pp. 219-249.
[14] G. Rothermel, M. J. Harrold, J. Ostrin and C. Hong, "An Empirical Study of the Effects of Minimization on the Fault Detection Capabilities of Test Suites", Proceedings of the International conference, On Software Maintenance, IEEE Computer Society, **(1998)**.
[15] T. Y. Chen and 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, **(1995)** April, pp. 415-424.