TOOL SUPPORT FOR TEST CASE SELECTION IN REGRESSION TESTING

Sujata[1] , G.N.Purohit[2]

[1]Department of Computer Science and Engineering, ITM University, Gurgaon, India
sujata@itmindia.edu
[2]Banasthali Vidhyapith, Banasthali, India
gn_purohitjaipur@yahoo.co.in

ABSTRACT

Software testing and retesting occurs continuously during the software development lifecycle to detect errors as early as possible. As the software evolves the size of test suites also grows. Due to limited resources, basic problem in regression testing is to select the important test cases from the available test suit. In this test case selection techniques unimportant test cases are discarded to save the time and resources. A number of techniques have been proposed for test case selection. Most of them are code based. So, they have not proved viable in all the situations. In this article we present an approach based on specifications. It uses classification tree method using classification tree editor tool. The classification-tree method provides a systematic way for software testers to derive test cases by considering important relevant aspects that are identified from the specification. The method has been used in many real-life applications and shown to be effective. The proposed approach is very effective to reduce manual effort in the generation and selection of test cases. We are using CTE XL tool for supporting this enhanced process which shows how to integrate weighting factors into classification trees which leads to automatic generation of selected test suites. CTE XL supports test case selection by occurrence probability, error probability, or risk because it is based on user requirements.

KEYWORDS

Regression Testing, Test Case Selection, Classification Tree Method, Fault Severity

1. INTRODUCTION

Regression testing is vital for ensuring software quality. It is the process of validating modified software to ensure that the changed parts of the software behave as intended and that the unchanged parts of the software have not been affected by the new modification[6,7]. A number of techniques are present in literature, but most of them are code-based[3]. Only a few techniques are requirement or specification based. Code-based regression test selection is good for unit testing, but it has a **scalability** problem. As the size of the system under test grows, it becomes harder to implement these techniques and to create corresponding traceability matrices. Also these techniques are time consuming and error-prone as there is need of complete involvement of the tester to access and understand code which is language-dependent.

Also specification based techniques are not always good as there are too many personal decisions involved in it. Testers, managers or other people involved in software testing follow their own criteria and there are no predefined standards for it. So we are providing a new solution to define a new regression test selection method which is based on tool support so that regression test selection criteria can be more objective.

This paper presents a new specification based test case selection technique which will use classification tree editor tool for the test case generation. We use requirement analysis to guide

DOI : 10.5121/ijsea.2011.2403

test case selection, and measure the quality of the regression test suite. This approach is an enhancement to the classification-tree method that further reduces the manual effort in the test case generation and selection of test cases.

2. RELATED WORK

Software testing is one of the activities of software development life cycle in which few resources are invested. Time provided for testing of the software is not adequate, so the challenges to be dealt by the real world testers have increased tremendously. Their resources are limited, deadlines arrive quickly, and the testing of the system crosses the time limit, hence it has no worth.

Testers has to take into consideration each and every possibility like test cases from existing projects can be considered or if there is lack of time then prioritization and selection of original test suit can be done. Even then there is no guarantee that the selected test suit is composed of important test cases. Also there is no way to identify the important test cases.

2.1. Category Partition Method

Quality of testing depends upon test case design activity which further determines the nature and scope of any test. If the test cases are determined on the basis of a specification, this is called a functional test. Although the functional test is of great importance for the verification of systems and is commonly used in industry, only few methods and tools are available for the systematic generation of corresponding test cases. This is why, in many cases, only partially applicable tools such as decision tables or MS Excel are used for test case determination.

The category partition method was developed by Ostrand and Balcer which provides stepwise guidelines that lead to the production of test cases and test scripts from the functional specification. Tester analyses the functional units to identify the categories of items of information that may affect the behaviour of the system. The relations and constraints among choices from various categories are characterised and expressed as a formal test specification in a test specification language (TSL) in textual format. The test specification is then automatically processed by a generator tool to produce a set of test frames. A test frame is a combination of choices which together form the description of a test case. The tester has to review these test case descriptions (which are produced by the generator tool) one by one, and may decide to revise the test specification in order to eliminate impossible choice combinations or to reduce the number of test cases to a manageable amount. These steps are iterated until the tester is satisfied with the test specification. Finally, the resulting test cases are organised by the tester into test scripts for execution

2.2. Classification Tree Editor

The classification-tree method is an efficient testing method for the functional test which was introduced by Grochtmann and Grimm in 1993. This method enables the division of the whole input domain of a test object into independent classes via classifications[5] as shown in figure1[8].

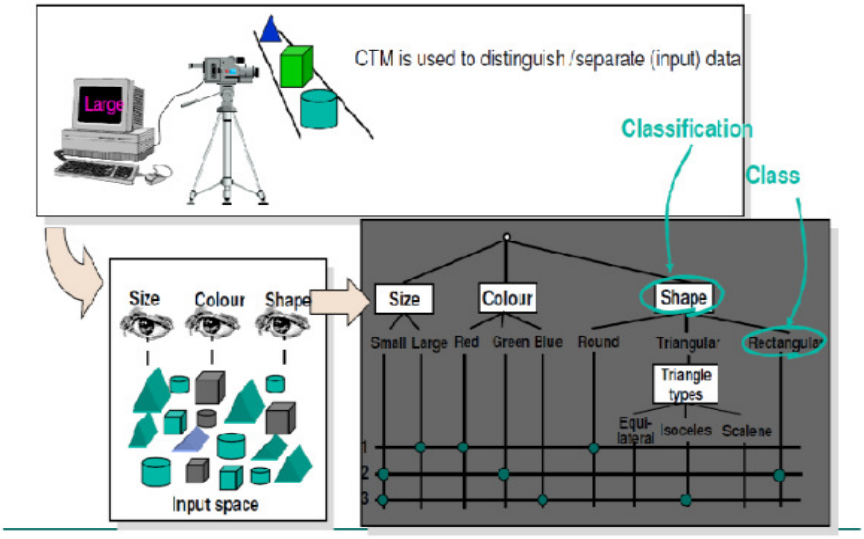

Fig 1 The Classification Tree Method [8]

The basic idea behind this approach is to first divide up the amount of possible inputs for the test object. It takes each and every aspects into consideration. The divided inputs are then combined to produce non-redundant test cases, which cover the entire input data domain.

Initially tester is required to identify attributes which are relevant to the test. Each attribute should allow a narrowly restricted and therefore clear differentiation between the possible inputs for the test object. So, the amount of possible inputs is divided up according to each attribute.

Mathematically it can be said that the division of the amount of possible inputs is disjoint and complete and they are divided into subsets, called classes. Thus one classification results for each attribute. The constraints among the classifications and classes are organised in the form of a hierarchical structure called the classification tree.

It is used as the start of a combination table in which the tester selects the combinations of classes that form the actual test cases to be used. In this the tester also needs to manually verify, one by one, that each selected combination of classes is logically compatible and therefore forms a valid test case. Invalid test cases have to be manually determined and removed.

Both CPM and CTM require the specification analysis to determine important aspects that are taken as the base for partitioning the inputs. In CPM, each aspect corresponds to a category partitioned into choices; whereas in CTM, the corresponding entities are named classification and classes. Let us now use the example presented in to illustrate these essential concepts.

The classification tree editor (CTE) has been created to help use classification-tree method more efficiently. It is a syntax-controlled, graphical editor, which offers productive support for test case determination using the classification-tree method. CTE assists the tester to organise the classification tree and interactively mark the selected test cases one by one. Although CTE allows text comments to be added to the classification tree, they serve primarily as labels for documentation and are not processed. CTE makes no distinction between test case generation and selection, as the tester performs these tasks manually while marking the classes to be combined.

All possible improvements became clear during the use of the CTE in many industrial software development projects. This has led to the development of new and revised version of CTE into CTE XL (Classification Tree Editor eXtended Logics) which is written entirely in Java. A screenshot of CTE XL is provided in Figure 3. Current versions of the CTE XL support automated test case generation, user-defined dependency rules, and the integration of requirement and test management tools.

Current test case generation offers four different modes:

1. *Minimal combination* creates a test suite which uses every class from each classification at least once in a test case.
2. *Pairwise combination* creates a test suite that uses every class pair from disjunctive classifications at least once in a test case.
3. *Threewise combination* ("triple-wise") creates a test suite that uses every triple of classes from disjunctive classifications at least once in a test case.
4. *Complete combination* creates a test suite that uses every possible combination of classes from disjunctive classification in a test case.

In addition to many improvements the CTE XL supports the specification of logical dependencies among classes of the classification tree. This has made it possible to reduce the amount of possible test cases assembled by the classification tree by more than 90% in individual cases. This aspect we are using in this paper as a test case selection technique. Besides the implicit logical rules, which the classification-tree method already provides, the CTL XL checks on these dependencies continuously during test case definition. This prevents the combination of contradictory classes within test cases on the one hand, and, on the other, allows test cases to be completed automatically, if it is possible to clearly deduce the choice of further classes from the classes which the user has selected in a test case and the existing dependency rules.

3. TEST CASE PRIORITIZATION USING CLASSIFICATION TREE EDITOR

The classification-tree method works in two main steps supported by the tool CTE XL: 1) design of a classification tree and 2) definition of test cases in the table. We used the systematic and efficient test case determination feature of the tool for our problem.

Our aim was to classify an amount of building blocks by size, color and form and then perform test case selection for the given problem. For this purpose, we generated a classification tree containing three classifications: size, color and shape. Classes such as small, large, red, blue, green, triangular, round, rectangular were assigned to these classifications. Using the test case generator, we generated the test cases for the given problem. The classification tree is visible in the drawing area for the classification tree and the test cases were shown in the test case list section of the tool. Situated beneath, is the combination table which provided the links of the test cases with the tree elements.

Test case were generated using Test case Generator in which we defined the condition that as many test cases should be generated as were required to combine all the classes of the **size** classification with all the classes of the **color** and **shape** classification. The rule was defined as **size*color*shape**. These test cases were present in a test suite **SiShaCo.** In Figure 2 we can see all the possible test cases generated after the specification of the rule.

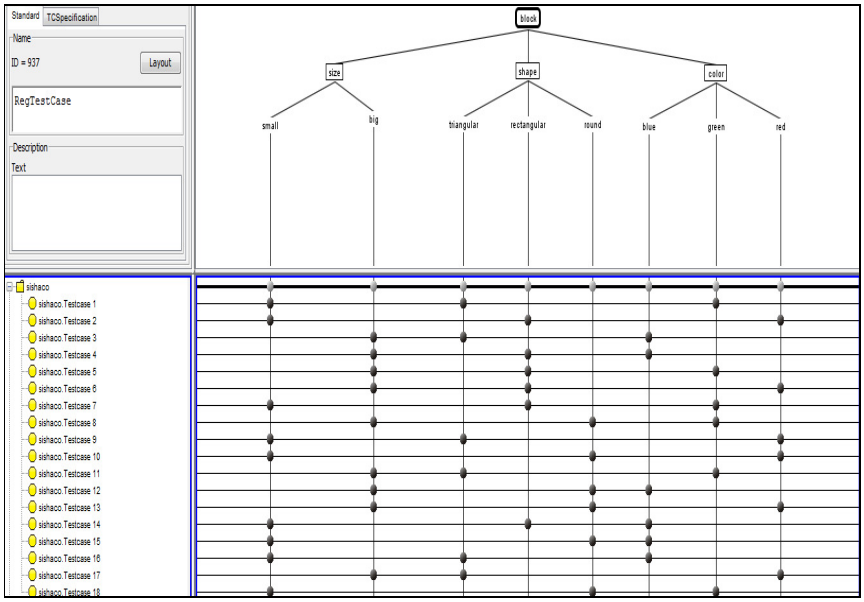

Fig 2 Test Cases generation

In the next step the test cases selection was performed. We need to be ensured that all the blocks should be of small size and of all the shapes and color. For this we defined the rule **Rule1** in the Dependency Editor and the rule was:

Rule1: (((((((((small AND triangular) AND blue) OR((small AND round) AND blue)) OR ((small AND rectangular) AND blue) OR ((small AND triangular) AND red) OR((small AND round) AND red)OR((small AND rectangular) AND red) OR((small AND triangular) AND green)OR((small AND round) AND green) OR((small AND rectangular) AND green) ;

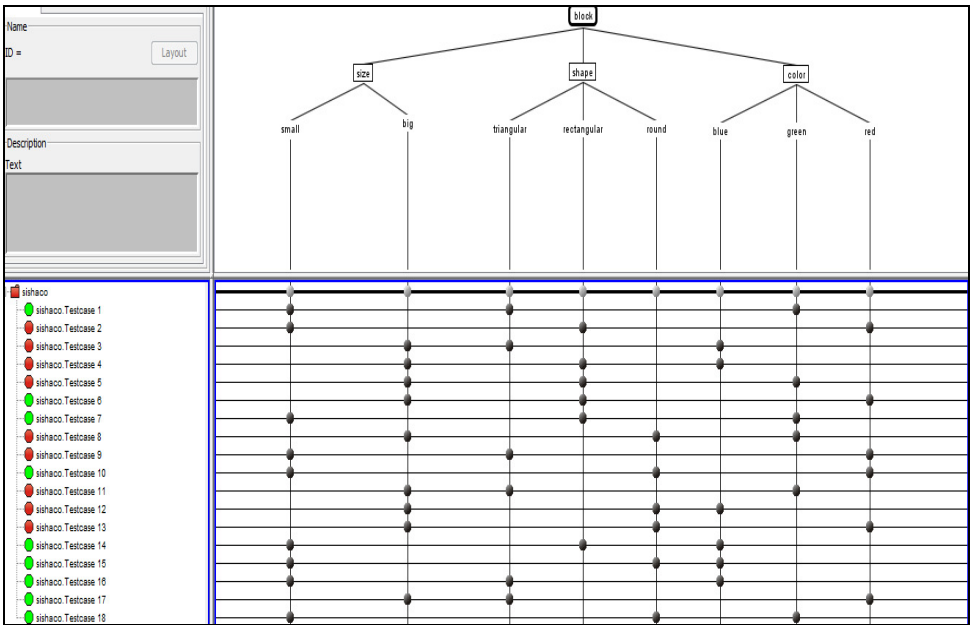

Fig 3 Marking of the Test cases

In Figure3, the marking of all those test cases takes place those were violating the rule. With the help of Dependency Manager we found that all those test cases that were obeying the rule were marked green and the one violating the rule were marked red. After this, again we generated the test cases with the help of the Test cases Generator, specifying the same condition for test case generation that is **size*color*form**. Now when the Test case Generator generated the test cases, it automatically followed the rule while generating the test cases. The test cases generated were kept in a new test suite **RegTest.**

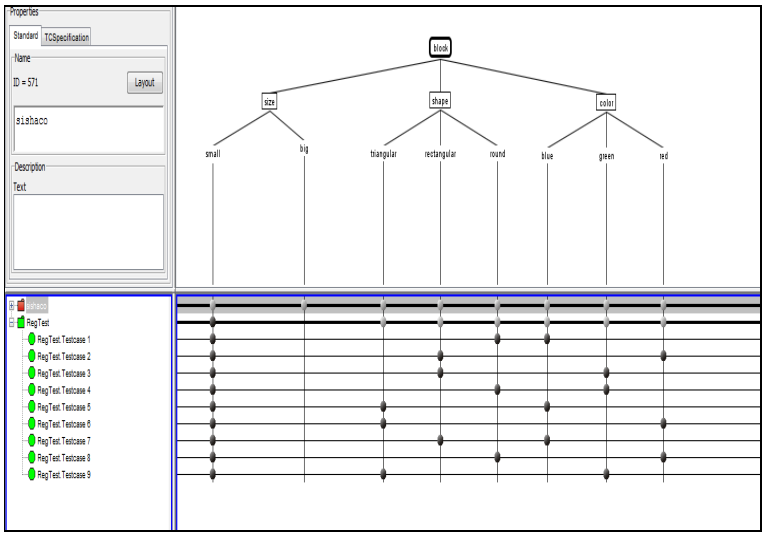

Fig. 4 Test Case Selection by rule

In Figure4, we had those test cases that were selected after the specification of the rule. The test case selection takes place which were place in the new test suite called **RegTest.** After the specification of the rule, in the dependency editor, test cases were generated and were placed in the **RegTest** suite. We observed that the number of test cases reduced. We made the use of the dependency manager to specify our requirement and then made the use of test case generator which generated all those test cases which satisfied the rule.

Also it is possible to perform the test case selection with the help of test case generator. But, using the Dependency Editor the logical errors are checked simultaneously.

CONCLUSION

CTM has provided the tester with a structured framework and methodical guidelines. In this paper, we proposed that the classification tree be annotated with additional information, namely, dependency editor. We have also proposed an improved, streamlined and highly automated process of preparing a selected test suite from an annotated classification tree. Within this process, the tester can concentrate his/her effort on making high level judgments and decisions such as the criteria for determining the legitimacy of test cases, and for selecting the test cases. The proposed technique focus on quality testing as it reduce the number of test cases which lead to the limited period of time and also minimum cost of test run.

REFERENCES

[1] David Wilmore, Suzanne M.Embury, (2005), "A Safe regression test selection technique for database-driven application", Proceeding of the 21st *IEEE International Conference on Software Maintenance(ICSM)*Lee, S.hyun. & Kim Mi Na, (2008) "This is my paper", ABC *Transactions on ECE*, Vol. 10, No. 5, pp120-122.

[2] Henry Muccini, (2007), "Using Model Differencing for Architecture-level Regression Testing", *33rd EUROMICRO Conference on Software Engineering and advanced Applications*.

[3] Gaurav Duggal, Mrs. Bharti Suri, (2008), "Understanding regression testing technique", *Proceedings of 2nd National Conference on Challenges and Oppotunities in Information Technology*.

[4] Liela Naslavsky, Hadar Ziv, Debra J. Richardson, (2009), "A Model based Regression Test Selection Technique", *Proceeding of IEEE International Conference on Software Maintenance(ICSM).*

[5] Matthias Grochtmann, Klaus Grimm, Joachim Wegener, (1993),"Tool-Supported Test Case Design for Black-Box Testing by Means of the Classification-Tree Editor", *EuroSTAR*' Vol. 93, 25 - 28.

[6] Michael Ruth, Feng Lin, Shengru Tu[2006], "Applying Safe regression Techniques to Java Web Services", *International Journal of Web Services Practices*, Vol.2, No.1-2 , pp. 1-10.

[7] Wong, et al.,(1998), "A study of effective regression testing in practice*", In Proceedings of 8th International Symposium on Software Reliability Engineering(ISSRE'97)*, pp. 264-274.

Authors

Sujata
M.Tech. (Software Engineering), M.Sc. (Computer Science)
Assistant Professor (ITM University, Gurgaon)

She is pusuing PhD from Banasthali Vidhyapith, Banasthali under the guidance of Prof. G.N. Purohit(Dean(AIM & ACT), Banasthali University, India). She did her M.Tech. from M.D.University . She did her M.Sc. from Kurukshetra University. She has 5 years of experience including teaching and industrial as a Software Tester. Her area of specialization are Software Unified Modelling Language, Software Quality and Testing.