ISSN:0975-9646

A NOVEL APPROACH FOR TEST CASE PRIORITIZATION USING PRIORITY LEVEL TECHNIQUE

Arup Abhinna Acharya#1, Goutam Budha#2, Namita Panda#3

#School of Computer Engineering, KIIT University,
Bhubaneswar-751024,Orissa, India

Abstract— **This paper presents a novel approach for test case prioritization using a simple mathematical prioritization level technique. In this we have identified a number of generic parameters under GUI, Database, Networking heads and taken into consideration a number of projects falling in a particular category of projects .For each of these projects falling under a given category, we utilize experts opinion to classify the level of user requirement concerning the parameters and the extent to which it has been satisfied by each of the projects, at the first instance, on a six scale basis. The information of all the tables is combined to generate a Project Specific Base Table (PSBT). Whenever a new project under the same category comes up, respective priority levels are assigned to each of these identified parameters, i.e. we prioritize the test cases concerning the identified parameters by utilizing the information from the PSBT and using some mathematical calculations.**

Keywords— **Prioritization, Project Specific Base Table.**

I. INTRODUCTION

For every software product being developed a considerable amount of time is to be spent on testing the product prior to its release and commercialization so that all the latent defects in the software are eliminated and the software behaves normally as per the expectation of the users using or interacting with the system in question. The reliability of software is a highly relative term with respect to the users. It may so happen that the software developed has got say for example five error prone functions out of a total of ten functions and say a user X when using the software invokes three error prone functions and let's say other users Y and Z invoke five error prone functions and no error prone function respectively. In that case Y would term the software as highly unreliable, X would term or classify the software as slightly unreliable and Z would classify the software as highly reliable. Thus from this example it is evident that software reliability is a very relative term based on the number of error prone functions being invoked. Thus while developing a software it is highly essential to ensure that it is error free and equally reliable for all the users interacting or using the given software. It is essential to eliminate this concept of relative reliability and ensure that the software behaves in an equally reliable manner for all the users. Thus for this Software Testing has a major role to play. Software Testing is an important phase of quality control in Software development and is necessary to produce highly reliable systems. According to IEEE testing may be defined as "**the process of exercising or evaluating a system or system component by manual or automated means to verify that it satisfies specified requirements or to identify differences between expected and actual results".**

In the present day scenario model-driven software development has evolved as a new paradigm. In this approach the developers use model based software testing for generating test cases for those software whose foundation rests on Object Oriented Principles. In contrast to traditional approaches Model Based Testing as is implied by its name is the generation of test cases and its analysis from design and analysis models which is also termed as Grey Box approach of testing. Since the designing phase is prior to the coding phase, the test case generation from the design models would avoid the blocking states which may otherwise be encountered by the testing team of the software development team in case of code based testing approach in which the testing phase can begin only after the successful completion of the coding phase. The testing and coding can be carried out simultaneously and many problems in the design can be uncovered even before the software is implemented. Thus this was a brief discussion about software reliability and the importance of testing in improving the reliability of the software. We also discussed about the importance of Model Based Testing as a new paradigm of software testing.

Now we turn our focus towards Prioritization of Test Cases. The major aspect of software testing is test case generation. To start with, a test case can be defined as a triplet [I,S,O] , where I is the input data given to the system, S is the state of the system in question and O is the output of the system [1]. A test suite which is the set of all the test cases must be optimal i.e. it must be of reasonable size and should be able to uncover maximum errors existing in the given system. Care must be taken that the test cases which constitute a given test suite should not be redundant as a result of which the size of the test suite becomes large with a decline in the performance. The concept of reusability is also applicable to the Testing phase of software development as the software developers often save the test suite for their software so that they can use these test suites for testing as the software evolves in due course of time. [2] This is basically what is the idea behind Regression Testing. Regression Testing can be defined as the testing activity which is carried out to ensure that no new bugs have been introduced in the software due to some changes made in the original version of the software or because of an attempt to fix some bugs in the software. Running all the test cases in the test suite is of course essential to ensure the overall reliability of the software but it is also essential to order the test cases so that they are run according to some given priority according to some criterion. This is test case prioritization which schedules the test cases to maximize some objective function. But the concept of Prioritization is not restricted to Regression Testing alone. The following points summarize in brief the various possible goals for prioritization:

- To increase the code coverage in a system at a faster rate, in the system under test so that a code coverage criterion is met at an early stage in the process of testing.
- To hasten their confidence regarding the reliability of the system at a faster rate.
- To detect the high risk faults in the system at a faster rate that too at an early stage in the testing process.
- To increase the rate of fault detection of the test suite designed for the system under test.

The objective we have identified for the prioritization of test cases as a motivation of our work in this paper is to maximize the level of satisfaction of the user of the software and also to increase the reliability of the system. In this paper we propose an approach in which we identify certain generic parameters based on which test cases can be designed. We need to analyze the user requirements for a number of projects say P1 to Pn where these projects fall in a particular category of project say a banking project. For each of these projects we propose to build up a table indicating the level of user requirement and the level to which the requirement has been satisfied by the software on the first instance of development(i.e. as soon as the software is ready without making an attempt to improve the achieved level of satisfaction to near the original level of requirement) on a 6 scale basis against the identified generic testing parameters. We then derive a common table from the above tables which we term as **PSBT** i.e. Project Specific Base Table which would indicate the overall level of user requirement on the 6 scale basis against the identified generic parameters. For a Project Pn+1 which comes up and falls in the same category of the project we propose to build up another table capturing the new level of user requirement and the prioritization factor of the test cases for each of the identified testing parameters through some proposed mapping technique from the **PSBT**. Although the approach of prioritization would be the same but the PSBT would be different for different category of projects and would be common for all the projects falling under a particular category.

The rest of the paper is organized as follows. Related Work is discussed in Section II. In Section III we present a thorough discussion of our approach followed by Conclusion and Future Work in Section IV.

II. Related Work

Rothermel et al. [3] investigated several prioritizing techniques such as total statement (or branch) coverage prioritization and additional statement (or branch) coverage prioritization that can improve the rate of fault detection. Coverage-based TCP techniques [4, 5, 6] involve ranking test cases based on the statement coverage they provide. Test cases are ranked based on the number of statements executed/covered by the test case such that the more lines of code the test executes, the earlier in the test cycle the test is run.

Ten coverage-based prioritization strategies [7, 4, 5, 6] are summarized below:

- *Random prioritization*: As is evident from the name, in this kind of prioritization, the ordering of test cases for execution is random.
- *Optimal prioritization*: Optimal prioritization technique is a theoretical technique that goes for ranking the test cases based on the number of faults they expose. In this approach it is assumed that the program faults are given as input and this information is used to iteratively select the test case that exposes the largest number of faults not yet exposed by already-selected test cases until test cases that expose all faults have been chosen
- *Total statement coverage and additional statement coverage prioritization*: In this strategy the test cases are ranked based on the number of statements executed or covered by the test cases in a way such that the test case covering the maximum number of statements would be executed first. Total statement coverage prioritization schedules test cases based on the total statements each test case executes and there is a likelihood that the same set of statements being covered by multiple test cases. On the contrary the Additional Statement Coverage Prioritization first selects the test case with maximum statement coverage followed by adjusting the coverage information on the remaining test cases to reflect the test cases not covered by that test case and iteratively selects a test case that provides the largest additional statement coverage until all program

statements have been covered by at least one test case.

- *Total branch coverage and additional branch coverage prioritization*: The test coverage is measured based on the program branches covered. Branch coverage is defined as covering each possible outcome of a condition. Additional branch coverage bears close resemblance to additional statement coverage except for the fact that it relies on branch coverage instead of statement coverage.

- *Total function coverage and additional function coverage prioritization*: Function coverage measures the total number of functions covered by the test case. This strategy bears close resemblance to total statement coverage with a difference that it measures coverage based on the total number of functions executed. Similarly additional function coverage prioritization differs from additional statement coverage prioritization in a way that prioritization takes place at the function level instead of the statement level.

The Average Percentage of Faults Detected (APFD) metric [5, 6] measures the benefits of code coverage based test case prioritization strategies. In a case study [5] conducted at Siemens Corporate Research Center to measure the effectiveness of TCP in improving the rate of fault detection and to compare different coverage-based prioritization techniques to measure their efficiency, the researchers used various prioritization techniques to measure the APFD values and found statistically significant results that APFD values were not the same for all of the techniques. The code coverage-based TCP strategies were shown to improve the rate of fault detection, allowing the testing team to start debugging activities earlier in the software process and resulting in faster software release than otherwise possible.
To talk of Requirements Traceability, Gotel and Finkelstein define it as the ability to describe and follow the life of a requirement from its origin to development to deployment in an iterative way [8]. Ramesh and Jarke describe RT as a quality attribute that is essential for a system to possess [9] in order to have good quality. Tahat discusses traceability as a mapping between requirements and test. If the test cases are not associated with individual requirements it could be difficult for testers to determine if the requirement is adequately tested [10] or in other words it is evident that if our test cases are associated with individual requirements the user satisfiability test can be performed very accurately thereby increasing the reliability of the software. Ramesh and Jarke highlight the importance of traceability to develop the system Compliance Verification Procedure to ensure that the system complies with the specified requirements and that it meets the needs of the users. [9] Srikanth et.al have proposed a value driven approach for system-level test case prioritization called the Prioritization of Requirements for Test (PORT). PORT prioritizes system test cases based upon four factors: requirements volatility, customer priority, implementation complexity, and fault proneness of the requirements. They have satisfied two major objectives through their proposed methodology: improve user perceived software quality in a cost effective way by considering potential defect severity and (2) to improve the rate of detection of severe faults during system level testing of new code and regression testing of existing code. To achieve the goal of early fault detection in the regression testing process, they have made an attempt to prioritize the test cases by considering relevant slice of the program, which comprises of those set of statements that influence or have got the capability to influence the output of a program when run on that test case. They propound a concept that if a modification in the program has an effect on the output of a test case in the regression test suite, it must affect some computation in the *relevant slice* of the output for that test case. J. M. Kim and A. Porter propose a test case selection technique suitable for long run of regression testing in constrained environments (authors mentioned their approach as prioritization techniques). Krishnamurthy et. al discuss an approach for Regression Test Case prioritization using Genetic Algorithms [11]. Daengdez et.al discuss various prioritization approaches [12]. Jiang et. al discuss Adaptive Random Test Case Prioritization Techniques [13].

III. Proposed Approach

Now we focus our attention on the approach we have proposed for the prioritization of test cases. In our approach we basically suggest a prioritization level for the test cases. The major elements actively involved playing a role in determining the prioritization level of the test cases are as follows:

- The SRS document
- The experts having an in depth experience in developing a particular category of projects say a banking project.
- The software developers.

In our approach the first step we do is to identify project independent generic parameters for testing which we need to develop the test cases. Each such parameter may have more than one test case which would be needed to be executed so as to successfully test that the given parameter component of the project is running is functioning without any latent errors. We categorize these parameters into three broad categories:

- GUI or the Graphical User Interface Parameters
- Database Parameters
- Networking Parameters

Under each category of the aforementioned parameters we have a number of identified parameters as well. We enlist these parameters as follows:

<u>GUI Parameters:</u>

- Degree of using intuitive command names
- Speed of use/Productivity support
- Degree of intellisense support
- In case of event triggered activities degree of confirmation of the events in the software
- Feedback from the system showing system status in case of complex event triggering
- Flexibility of help (online/ offline)
- Degree of error recovery
- The extent of jargon free error messages to be displayed
- Overuse or underuse of modal dialogs
- If there is a sequence of modal dialogs being used in response to an event
- Degree of mapping between use case and User Interface design
- User flexibility in terms of hybrid use of Command Line Interfaces, Iconic and Graphical User Interfaces.
- Degree of mapping in case of use case and UI design
- Use of primitive or highly composite commands in case of the software deploying command line interface (CLI).
- Degree of component based nature of the software.

Database Parameters:

- The equivalence of a schedule to a serial schedule in case view serializability/conflict serializability is followed.
- Access time from interface
- The degree of successful recovery of the database from the backup.
- Speedup or scaleup in case of parallel database and to increase the speedup
- Level of normalization
- Degree of successful recovery in case of recoverable schedule being followed.
- Degree of effectiveness of data accessibility in case of Distributed Database System is being used for the application
- Level of security to the database of the application

Networking Parameters:

- Level of network security
- Efficiency of the physical topology being used
- Reliability of the communication channel
- Time of communication
- Efficiency of the Network model being deployed

Any given project is not limited only to these set of identified parameters. There may be some extra identified parameters which may be added on by the development team during the course of requirement analysis. Also a project may not be using all of the above identified parameters. We present below a block diagram communicating our overall idea.

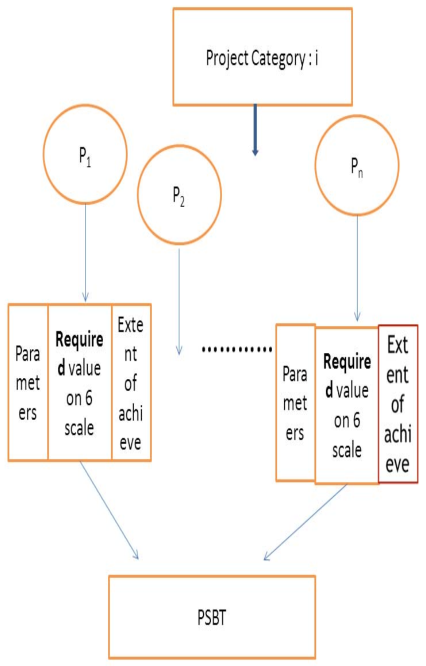

Fig. 1 Phase 1 of our proposed model

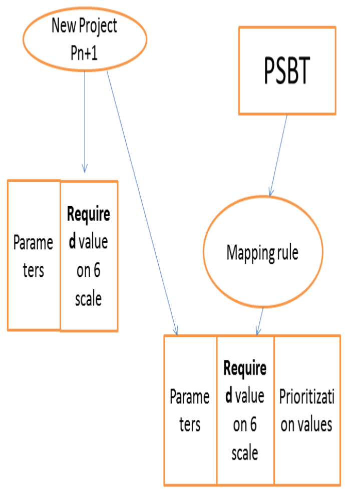

Fig. 2 Phase 2 of our proposed model

The above represented diagram conveys the information, that we identify a particular category of project at first. The category of project may be for example say a banking project, a satellite launching project or say a credit card development project, etc.

Let us take for example that Project Category i corresponds to the banking project. The banking projects P_1 to P_n for various banks may be developed by one or many organizations. For each of these projects we populate a table having three columns (Parameters identified, Required Value indicating the customer requirement/ system requirement on a six scale basis, Extent of achievement indicating the extent of achievement of the requirement at the first instance of development of the project). We collect such data from P_1 to P_n projects and finally come up with PSBT which stands for Project Specific Base Table as the outcome of the first phase of Prioritization. The PSBT has got the same structure as that of the tables for individual projects. For each parameter we collect the values from the tables P_1 to P_n corresponding to each column and find their arithmetic mean. The PSBT is populated with the arithmetic mean (calculated from the tables P_1 to P_n) against each parameter.

$$\text{Value}(i,j) = \left(\sum_{i=1}^{n} P_i \right) / n \qquad (1)$$

where, i is the row corresponding to a particular parameter , j corresponds to one of the last two columns of the PSBT and n is the number of the historical tables we have.

Supposing that a parameter identified is not relevant to the given category of project the value in each of the columns against the parameter would be assigned as zero. As already mentioned the list of identified parameters is an illustrative list and each of the identified parameter lists are subject to further addition of parameters. The parameters we have identified are very generic to almost all the category of projects and are likely to be used in almost all the category of projects.

Given a project P_1 falling under a particular category say Category X, it is the combined duty of the system analyst and the experienced developers to use their expertise in analyzing the customer requirements and assigning a value based on a 6 scale basis under the **Required Value** column against each parameter prior to the development of the project. There is a likelihood that the identified parameters are too technical to be specified by the customer or the prospective user of the software. Or in other words if there exists any such parameter which is so system specific that it is to be taken care of by the developer developing the system then in that case the developer will take the responsibility of assigning the values against the parameter in question. Similarly the developers, test engineers, system analysts have to go for a combined effort in populating the **Extent of achievement** column with a value on a 6 scale basis denoting the level to which the incorporation of the requirement denoted by the parameter has been satisfied in the first instance just after the completion of the project i.e. if after the initial system testing it is found that a parameter P having the level of requirement 4 has been rated as 3.5 as the satisfaction level then 3.5 is the value to be considered. A similar methodology is to be used for populating all the tables P_2 to P_n falling under the Category X. Finally we come up with the PSBT as the outcome of the first phase where the values in the PSBT are generated as per (1). The PSBT is common for a given category of project. In the second phase when a new project P_{n+1} comes up we also come up with a table having three columns namely the identified parameters, the rating value assigned by the expert against the parameter on a 6 scale basis and the prioritization level of the test case for the parameter (which is initially not populated but using a mapping algorithm we populate the column taking into account the Project Specific Base Table or PSBT). Thus we propose the concept of using historical data for a particular category of project available in the form of PSBT in assigning priority values to the test cases in a new project falling in the same category of project. The mapping algorithm we talk of uses the concept of simple mathematics as an aid in assigning priority values. Now we focus our attention on the algorithm devised to achieve the objective.

We achieve our objective in two phases:

- In Phase 1 we go for constructing the PSBT
- In phase 2 we go for constructing two Intermediate Tables followed by final assignment of priority levels

At the first instance we go for populating the **Required Value** and the **Extent of Achievement** columns of all the tables P_1 to P_n belonging to the same category of the project. We now present a small algorithm for populating the PSBT with values based on (1).

```
Algorithm Construct PSBT
Input: Tables P_1 to P_n
Output: PSBT

1.   For every column j
     1.1. For every parameter k
          1.1.1. For every table n
                 1.1.1.1. Assign value of parameter k to x
                 1.1.1.2. sum=sum+x
          1.1.2. End For
          1.1.3. val= sum/n
          1.1.4. Assign column j of parameter k of PSBT with
                 the value val
     1.2. End For
2.   End For
```

After this we enter into phase 2 of our method. This involves the construction of two tables namely Intermediate Table 1 and Intermediate Table 2. The schema for Intermediate Table 1 is <Parameters, Extent of Achievement, New Requirement, Extent of Level of Satisfaction> and that for Intermediate Table 2 is <Parameters, Modified Level of Satisfaction>

In the Intermediate Table 1 Parameters column enlists the list of identified parameters, Extent of Achievement column takes the values from the corresponding column of PSBT, New Requirement takes values from the Level of Requirement column of the new project P_{n+1}, Expected Level of Satisfaction is the level to which the functionality of the identified parameter must be satisfied in the new Project. For example if for parameter X the Expected Level of Satisfaction of Intermediate Table 1/ Modified Expected Level of Satisfaction of Intermediate Table 2 has the value 3, it implies that the Expected Level/ Modified Expected Level must be strictly greater than 3. Intermediate Table 2 has got two columns namely Parameters enlisting the list of identified parameters and Modified Expected Level of Satisfaction which is calculated from the New Requirement column of Intermediate Table 1. We now present the algorithm for constructing Intermediate Table 1.

```
Algorithm Construct Intermediate Table 1
Input: PSBT
Output: Intermediate Table 1

1. Take the LCM of the Requirement column of PSBT
2. For every parameter n of PSBT
   2.1. Calculate factor = LCM / value of requirement
        column of PSBT
   2.2. Multiply Extent of Achievement by factor
   2.3. Collect the new requirement value from table $P_{n+1}$
        corresponding to parameter n
   2.4. Multiply the value collected in 2.3 with factor.
   2.5. Assign the values collected in 2.2 and 2.4 in the
        Extent of Achievement and New Requirement
        Column respectively.
   2.6. Calculate Expected Level of Satisfaction by the
        formula
        (LCM / value of level of satisfaction column ) *
        New Requirement.
3. End For
```

```
Algorithm Construct Intermediate Table 2
Input: Intermediate Table 1
Output: Intermediate Table 2

1. Take the LCM of the New Requirement column of
   Intermediate Table 1
2. For every parameter n
   2.1. Calculate factor = LCM / New Requirement Value
        Collected from Intermediate Table 1
   2.2. Multiply value of Expected Level of Requirement
        with factor to get Modified Expected Level of
        Satisfaction
3. End For
```

```
Algorithm Assign Priority
Input: Intermediate Table 2
Output: Final Table for Project $P_{n+1}$

1. Arrange the values of Modified Expected Level of
   Satisfaction in ascending order tracking the
   corresponding parameter.
2. Assign highest priority to the parameter having the lowest
   value of Modified Expected Level of Satisfaction.
```

Our reason of assigning highest priority to the parameter having the lowest value of Modified Expected Level of Satisfaction can be understood with a small illustration. Let's say we have the LCM of the New Requirement column of Intermediate Table 1 as 20 and Intermediate Table 2 as:

Intermediate Table 2 with dummy values

PARAMETERS	MODIFIED EXPECTED LEVEL OF SATISFACTION
X	12.5
Y	13.5
Z	10.0

We know from previous discussion that the values in the Modified Expected Level of Satisfaction column indicate that the level of satisfaction has to be strictly greater than the specified values. Thus if our LCM of the New Requirement column of Intermediate Table 1 is 20, since Modified Expected Level of Satisfaction is calculated keeping 20 as the base value we may infer that for Parameter X we need to implement the functionality in such a way so that our expected level goes above 12.5. Similar is the inference for the parameters Y and Z. Thus we need to do more work so as to achieve an expected level more than 10 than achieving an expected level more than 12.5 say. Although we must achieve more than the expected level of satisfaction for all the parameters enlisted, still increasing the target of achievement from 10 to a value lying in the close proximity of 20 should be given a higher priority than increasing the target of achievement from 13.5 to a value lying in the close proximity of 20. Thus we must execute the test case for Parameter Z followed by that of Parameter X and finally Parameter Y needs to be tested to find out their level of satisfaction. We believe that the satisfaction of the functionality of every parameter to a very close proximity of the required level is important for ensuring a high customer satisfaction.

IV. CONCLUSION

It has been already discussed how important is Prioritization of the test cases. In our approach we have relied on simple mathematics where we treat the satisfaction of every parameter equally important without discriminating between the parameters as we believe that to ensure the delivery of highly satisfactory software we need to take equal care of all the parameters. We were also theoretically able to justify our algorithm from the small illustration presented above. Our method studies the historical data of the projects falling under a particular category and attempts to mathematically utilize the data in aiding for assignment of prioritization level to the parameters of the new project at hand.

REFERENCES

[1] Pressman Roger S. , Software Engineering A Practitioners' Approach.

[2] K. Onoma, W.-T. Tsai, M. Poonawala, and H. Suganuma. Regression testing in an industrial environment. Comm. Of the ACM, 41(5):81–86, May 1988.

[3] S. Elbaum, A. G. Malishevsky, and G. Rothermel, "Test case prioritization: A family of empirical studies", IEEE Transactions on Software Engineering, 28(2), 2002 , pp.159–182.

[4] S. Elbaum, A. Malishevsky, and G. Rothermel, "Test Case Prioritization: A Family of Empirical Studies," IEEE Transactions on Software Engineering, vol. 28, pp. 159-182, February, 2002.

[5] G. Rothermel, R. Untch, C. Chu, and M. Harrold, "Test Case Prioritization," IEEE Transactions on Software Engineering, vol. 27, pp. 929-948, October, 2001

[6] G. Rothermel, R. Untch, C. Chu, and M. Harrold, "Test Case Prioritization: An Empirical Study," International Conference on Software Maintenance, Oxford, UK, pp. 179 - 188, September 1999.

[7] S. Elbaum, A. Malishevsky, and G. Rothermel, "Prioritizing Test Cases for Regression Testing," Proceedings of the ACM International Symposium on Software Testing and Analysis, vol. 25, pp. 102-112, August 2000.

[8] O. Gotel and A. Finkelstein, "An Analysis of the Requirements Traceability Problem," International Conference Requirements Engineering, pp. 94-101, 1994.

[9] R. Balasubramaniam and M. Jarke, "Toward Reference Models for Requirements Traceability," *IEEE Transactions on Software Engineering*, vol. 27, pp. 58-93, January 2001.

[10] L. Tahat, B. Vaysburg, B. Korel, and A. Bader, "Requirement-Based Automated Black-Box Test Generation," 25th Annual International Computer Software and Applications Conference, Chicago, Illinois, pp. 489-495, 2001.

[11] R. Krishnamurthy, S.A. Sahaya Arul Mary, "Regression Test Suite Prioritization using Genetic Algorithms", International Journal of Hybrid Information Technology, Vol.2, No. 3, July 2009.

[12] S. Roongruangsuwan, J. Daengdej, "Test Case Prioritization Techniques", Journal of Theoretical and Applied Information Technology, Vol. 18, No.2, pp. 45-60, 2010

[13] Bo Jiang, Zhenyu Zhang, W K Chan, T H Tse, "Adaptive Random Test Case Prioritization", Automated Software Engineering (2009)