World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:2, No:6, 2008

A New Type of Integration Error and its Influence on Integration Testing Techniques

P. Prema, and B. Ramadoss

Open Science Index, Computer and Information Engineering Vol:2, No:6, 2008 publications.waset.org/8525/pdf

***Abstract*—**Testing is an activity that is required both in the development and maintenance of the software development life cycle in which Integration Testing is an important activity. Integration testing is based on the specification and functionality of the software and thus could be called black-box testing technique. The purpose of integration testing is testing integration between software components. In function or system testing, the concern is with overall behavior and whether the software meets its functional specifications or performance characteristics or how well the software and hardware work together. This explains the importance and necessity of IT for which the emphasis is on interactions between modules and their interfaces. Software errors should be discovered early during IT to reduce the costs of correction. This paper introduces a new type of integration error, presenting an overview of Integration Testing techniques with comparison of each technique and also identifying which technique detects what type of error.

***Keywords*—**Integration Error, Integration Error Types, Integration Testing Techniques, Software Testing

I. INTRODUCTION

INTEGRATION Testing (IT) is an important part of the testing process in software industry. In function or system testing, the concern is with overall behavior and whether the software meets its functional specifications or performance characteristics or how well the software and hardware work together. This explains the importance and necessity of IT for which the emphasis is on interactions between modules and their interfaces. Test cases are specifically selected to test these interfaces rather than the functionality of the modules. Software errors should be discovered early during IT to reduce the costs of correction. Nowadays many organizations have found more benefit in building teams of developers and testers to perform IT [1]. It is aimed at exposing problems that possibly arise when two components are combined. Typical problems identified in IT are improper call or return sequences, inconsistent data validation criteria and inconsistent handling of data objects.

The goal [1] of IT is to put the units in their intended environment and exercise their interactions as completely as possible. Regardless of what approach is used for integration, incremental or otherwise, at some point during development, it is necessary to exercise the connections between units and it is useful to have one or more quantitative criteria to evaluate how well an interface has been exercised. Sometimes unit testing techniques are applied during IT that suffers two problems [2]. First, the unit testing techniques are usually too expensive to be practically applied during integration, and second, there is no reason to believe that they will find the kinds of faults that appear during integration. Some software faults cannot be detected during unit testing; these are often faults in the interfaces between units. Thus, specific tests must be designed to deduct integration faults. IT refers to testing interfaces between components to assure that they have consistent assumptions and communicate correctly [3].

As in [4] Chan and Chen presented an overview of research work on IT for object-oriented programs. Jin, Offutt [5] had applied coupling-based IT to moderately-sized software systems. The results were compared with the category-partition method on their effectiveness in detecting faults, which found that the coupling-based testing technique detected more faults with fewer test cases than category-partition.

Haley and Zweben [6] had identified and classified IE into two categories namely computational and domain IE. Later on, Delamaro, Maldonado, and Mathur [1] classified IE into three categories.

In section II a new type of integration error is introduced. Section III presents an overview of IT techniques. Section IV makes a comparison and discussion of IT techniques with regard to types of error detected. Section V presents the summary of this paper.

II. INTEGRATION ERROR

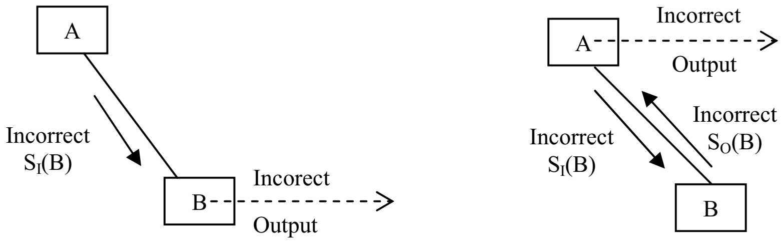
An error is a mistake of commission or omission that the developer makes. An error causes a defect. In software development one error may cause one or more defects in requirements, designs, programs, or tests. When an incorrect value is passed through a unit connection, then an Integration Error (IE) occurs.

Based on Haley et.al., [6] and Delamaro et. Al., [1] observations and further analysis, IE can be classified into four categories, thus introducing a new type of IE namely the *Type 4 error.* These categories are described as follows. Consider P as a program and t as a test case for P. Suppose that in P there are units A and B such that A makes calls to B. Let $S_I(B)$ be the n-tuple of values passed to B and $S_O(B)$ the n-tuple of values returned by B. When executing P on test case

P. Prema (Research Scholar) is with the Department of Computer Applications, National Institute of Technology, Tiruchirapalli, 620015, India (e-mail: mrgprem@ yahoo.com).

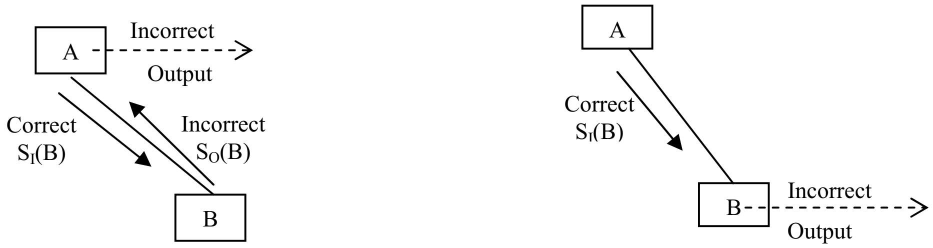
B. Ramadoss (Professor) is with the Department of Computer Applications, National Institute of Technology, Tiruchirapalli, 620015, India (e-mail: brama@nitt.edu).

Open Science Index, Computer and Information Engineering Vol:2, No:6, 2008 publications.waset.org/8525/pdf



(a) Type 1 error

(b) Type 2 error



(c) Type 3 error

(d) Type 4 error

Fig. 1 Types of IE

t, an IE is identified in a call to B from A when:

Type 1 error: Upon entering B, $S_I(B)$ does not have the expected values and these values cause an erroneous output (a failure) before returning from B.

Type 2 error: Upon entering B, $S_I(B)$ does not have the expected values and these values lead to an incorrect $S_O(B)$, which in turn causes an erroneous output (a failure) after returning from B.

Type 3 error: Upon entering B, $S_I(B)$ has the expected values, but incorrect values in $S_O(B)$ are produced inside B and these incorrect values influence an erroneous output (a failure) after returning from B.

Type 4 error: Upon entering B, $S_I(B)$ has the expected values and these values cause an erroneous output (a failure) before returning from B.

The above first three types do not specify the location of the fault responsible for causing incorrect outputs, they simply considers the existence of incorrect values entering or exiting a unit call, which is not so in the Type 4 error. In Type 4 error, when $S_I(B)$ has the expected values, a fault in B produces an erroneous output before returning from B. In this case, there is no error propagation through the connection A-B. This type of error is expected and to have already been detected during unit testing.

A Type 1 error occurs when an actual parameter or a global variable is passed from the calling unit incorrectly to the called unit and that unit produces an incorrect output. The flow in this case is shown in Fig. 1a. In a Type 2 error, there is an incorrect value entering the called unit and an incorrect value leaving that unit. This leads to an incorrect output in the calling unit (see Fig. 1b). A Type 3 error has one or more incorrect values leaving the called unit. In this case, a unit is called with correct input parameters but performs an incorrect computation which results in an incorrect return value which in turn leads to an incorrect output. This situation is illustrated in Fig. 1c. Type 4 error occurs, when an actual parameter or a global variable is passed from a calling unit to the called unit and that unit produces an incorrect output. The flow in this case is shown in Fig. 1d.

In Table I, the types of IE have been illustrated. In this if $S_I(B)$ and $S_O(B)$ are correct, output (B) are correct. This produces no error. If $S_I(B)$ is incorrect and $S_O(B)$ is correct then output(B) is correct, this situation should not happen in IT (shown in bold in Table I).

Finally, the n-tuples $S_I(B)$ and $S_O(B)$ depend partly on the program language. For example, in C language a unit is a function and n-tuples $S_I(B)$ and $S_O(B)$ can be defined as:

- $S_I(B)$: The n-tuple of input values in a call to a function B is determined by
  - the input parameters used in the function call and
  - the global variables used in B

- $S_O(B)$: The n-tuple of output values in a call to a function B is determined by
  - the output parameters used in the function call,
  - the global variables used in B, and
  - the values returned by B

Open Science Index, Computer and Information Engineering Vol:2, No:6, 2008 publications.waset.org/8525/pdf

TABLE I
TYPES OF IE

S.No	$S_I(B)$	$S_O(B)$	Output (B)	Type of Error
1	**C**	**C**	**C**	**no error**
2	C	C	I	4
3	C	I	I	4
4	C	I	C	3
5	**I**	**C**	**C**	**should not happen**
6	I	C	I	1
7	I	I	I	1
8	I	I	C	2

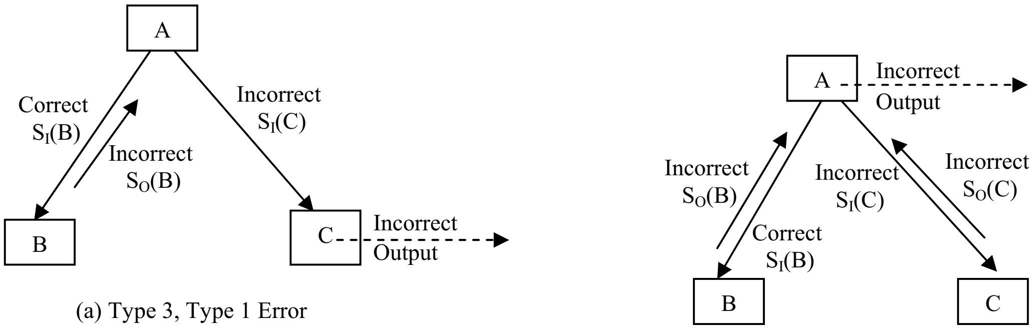
C: Correct I: Incorrect

In addition, more than one IE can be associated with (or caused by) a single fault. For example, consider a program P with three units A, B, and C such that A calls B and, upon returning from B, calls C. Suppose that in unit B sends an incorrect value x to C i.e., $x \in S_O(B)$ which is a part of $S_I(C)$. Suppose that due to x, C produces an incorrect output. Thus, a fault in B produced a Type 1 error in the connection A-C and a Type 3 error in the connection A-B (see Fig. 2a).

Similarly, in unit B return an incorrect value $x \in S_O(B)$ which is a part of $S_I(C)$ and due to x, C returns an incorrect output. Thus, a fault in B produced a Type 3 error in the connection A-B and a Type 2 error in the connection A-C (see Fig. 2b). If unit A sends an incorrect value x to B i.e., $x \in S_I(B)$, unit B return an incorrect value $x \in S_O(B)$ which is a part of $S_I(C)$. Suppose due to x, C produces an incorrect output. Thus, a fault in B produced a Type 2 error in the connection A-B and a Type 1 error in the connection A-C (see Fig. 2c).

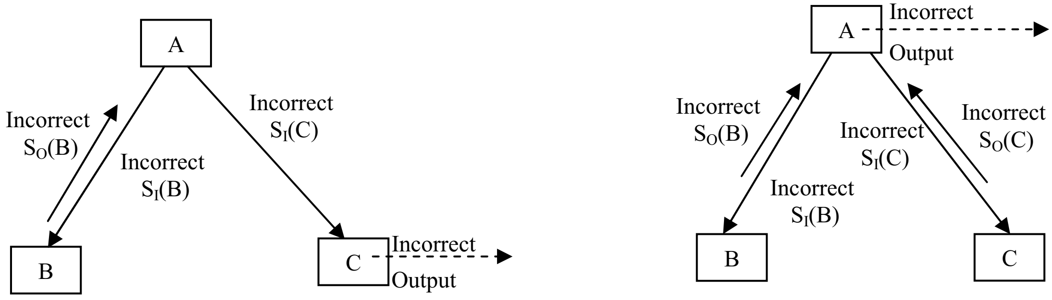
Similarly, if unit A send an incorrect value x to B i.e., $x \in S_I(B)$, unit B return an incorrect value $x \in S_O(B)$ which is a part of $S_I(C)$ and due to x, C returns an incorrect output. Thus, a fault in B produced a Type 2 error in the connection A-B and the connection A-C (see Fig. 2d). This single fault with more IE is given in Table II. In all other situations, B and C are independent. That is there is no relationship between B and C.

From the above situations, it is concluded that, if the value sent by A is correct or incorrect, it should produce only Type 1 or Type 2 error after returning from unit B. In this situation the value send to A should be incorrect. If the error is of Type 1, then we can identify the error early. Because before returning from that unit it shows the output as incorrect. But if the error is Type 2, it is difficult to identify the error before returning from that unit as only the errors can be identified when the program executes the result.



(a) Type 3, Type 1 Error

(b) Type 3, Type 2 Error



(c) Type 2, Type 1 Error

(d) Type 2 Error

Fig. 2 Single fault with more types of IE

Open Science Index, Computer and Information Engineering Vol:2, No:6, 2008 publications.waset.org/8525/pdf

TABLE II
SINGLE FAULT WITH MORE TYPE OF IE

S.No	$S_I(B)$	$S_O(B)$	Output (B)	$S_I(C)$	$S_O(C)$	Output (C)	Error Type
1	C	I	C	I	I	I	3, 1
2	C	I	C	I	C	I	3, 1
3	C	I	C	I	I	C	3, 2
4	I	I	C	I	I	I	2, 1
5	I	I	C	I	C	I	2, 1
6	I	I	C	I	I	C	2, 2

C: Correct I: Incorrect

III. APPROACHES FOR IT

This section reviews the following IT techniques namely Interface mutation based IT, coupling-based criteria for IT, data flow based IT, and Classification-Tree based IT.

A. Interface Mutation (IM) Based IT

IM is a mutation-based interprocedural criterion. It is an extension [1] of Mutation Testing (MT) and is applicable, by design, to software systems composed of interaction units. Both are powerful method for finding errors in software programs [7] and IM is used to evaluate how well the interactions between various units have been tested. The development of IM was motivated by the need to assess test sets for subsystems which come about due to the integration of two or more units. Applying IM, the syntactic changes are made only at the interface related points or connections between units. An IM operator is intended to mutate the program in ways analogous to the errors that may be committed by a programmer during program development. Once the mutants are generated, the next steps in IM are: to execute the mutants, to evaluated test set adequacy, and to decide mutant equivalence.

MT is complicated and time-consuming to perform without an automated tool. UNIX sort [1] utility was seeded with several integration errors and then tested with IM. This approach is used to reducing the cost of MT. Alternative IM criteria using different sets of IM operators were also evaluated. While comparing the error revealing effectiveness of these IM based test sets with same size randomly generated test sets, in most cases IM-based test sets are superior. The results suggest that IM offers viable test adequacy criteria for use at the integration level. PROTEUM/IM tool supports the application of IM criterion and exploration of alternative mutation criteria [8] at the IT phase.

B. Coupling-Based Criteria (CBC) for IT

Coupling [2] is a testing of connections between components during software integration. It provides the summary information about design and structure of the software and on the dataflow between the program units. Jin and Offult [2] had classified coupling between two units into twelve levels. These levels are not needed for testing, so it can be combined and classified into four unordered types: call coupling, parameter coupling, shared data coupling and external device coupling. CBC for IT requires that the program execute from definitions of actual parameters through calls to uses of the formal parameters. Therefore different coupling paths are defined. Coupling coverage analysis tool [3] can be used to support integration testing of software components.

Coupling [2] between two units measures the dependency relations between two units by reflecting the interactions between units. Faults in one unit may affect the coupled unit [9]. Each connection between program units is covered. These criteria have expected to be used both to guide the testers during IT. Coupling coverage analysis tool [3] can be used to support IT of software components, and satisfies part of the USA's Federal Aviation Authority's (FAA) requirements for structural coverage analysis of software.

C. Data-Flow (DF) based IT

DF testing [10, 11] has been used to test whether the program variables are appropriately created and used. Def-use pairs are determined by solving the data flow problem of reaching definitions. The testing of large programs usually takes place at several levels. The individual program units are tested first in isolation during unit testing. Then, their interfaces are tested during one or more integration steps [12]. Each step requires the computation of the def-use pairs that cross the most recently integrated procedure interfaces to establish the new test requirements. Exhaustively re-computing reaching definitions and def-use a pair at the beginning of each integration step is inefficient and may easily result in overly high analysis times.

Duesterwald et al., [13] defined Demand-Driven Analyzer (DDA) as a more efficient analysis approach for data flow based IT. They compared its performance of (i) a traditional exhaustive analyzer and (ii) an incremental analyzer. Demand-driven algorithm is the context of bottom-up IT. In the traditional analysis approach, the computation of data flow at one point requires data flow computations at all program points. It reduces the cost of IT through demand-driven analysis design. Incremental analysis avoids re-computation

Open Science Index, Computer and Information Engineering Vol:2, No:6, 2008 publications.waset.org/8525/pdf

TABLE III
COMPARISONS AND DISCUSSIONS OF IT TECHNIQUES

	IM	**DF**	**CBC**	**CT**
Overview	- White box and error based testing technique - IM is an extension of MT - In MT one takes a program and create many mutants by applying simple changes to the program - IM has been proposed as a criteria for the assessment of the adequacy of tests generated during IT	- static analysis for computing the def-use pairs - used to test whether the program variables are appropriately created and used - white-box testing technique	- provide information about design and structure of the software - goal is testing of connections between components during software integration - black-box testing technique - decide whether software has been adequately tested for a specific testing criteria	- Derive test cases from the specification - Based on the idea of partition testing - Black-box testing - Classification is the major entity of the system and sub entity is the class
Test Procedure	- To select and generate a set of mutants - To execute the mutants - To evaluate test set adequacy - To decide mutant equivalence	- test the individual program units - during integration step test the interfaces - select the sets of def-use pairs in a program - create test requirements	- map the parameter variables between units - apply data flow analysis to each unit - find the requirements of test cases and using dynamic analysis generate test cases	- Identify classifications - decompose them into equivalence classes -construct the classification-tree - generate test cases from the Classification-tree
Support Tool	- PROTEUM/IM	- Prototype testing tool	- Coupling coverage analysis tool	- CTE
Input	- Program, mutation operators, test data, test cases, variables	- Program, Def-use pairs, Test data execution path	- parameters/ arguments/ variables, all-coupling-uses criteria, choices	- test data
Output	- Mutants, Tested data, Test sets, Mutation score	- test case, def-use pairs, test requirements	- test sets, tested data	- tested data
Effectiveness	- compare the error revealing effectiveness of IM based test sets with same size randomly generated test sets, in most cases IM based test sets are superior	- it is faster than exhaustive analysis	- detect more faults with fewer test cases, - help the testers to find a rational, mathematical – based point at which to stop testing	- eliminate some invalid test cases
Effort / Cost	- Reduce cost	- reduce the cost, avoid the short comings of previously analysis approaches	- faults in one unit may affect the coupled unit	- To reduce number of test cases
Types of Error Detected	- High chance – 1, 2, 3 - Low chance – 4	- High chance – 1, 2 - Low chance – 3, 4	- High chance – 1, 3, 4 - Low chance – 2	- High chance – 1, 3 - Low chance – 2, 4

Open Science Index, Computer and Information Engineering Vol:2, No:6, 2008 publications.waset.org/8525/pdf

by performing the appropriate updates of a previously computed solution and also used in IT to extend the solution after each integration step with newly established reaching definition. It avoids the shortcomings of previously analysis approaches. Exhaustive information propagation is entirely avoided and replaced with a goal-oriented search. Unlike incremental analysis, using DDA for IT does not require the storage of reaching definition solutions between integration steps.

D. Classification-Tree (CT) based IT

In CT Method, disjoint and complete classifications are formed and represented as a tree. It is an extension of Category Partition method [14, 15] and both have been proposed for deriving test cases from the specification. These two methods are based on the idea of partition testing [16, 17]. The tree is used to derive test cases from the specification. CTM is based on the idea of partition testing. Classification is the major entity (main problem) of the system. Classification contains a set of classes that share a common structure and common behavior. The sub entity of the classification is the class, which represents some specific input value. Each class may be subdivided into subclasses. The terminal class (class without any subclasses) includes input data that can be used as test input. Classification can be used to identify the overall idea of the system. A test method [18] is used to identify test cases from the combination of system specification and COTS specification based on the CTM.

To identifying test cases [18], first form a classification-tree based on the system specification and then form another classification-tree based on the COTS specification. By overlapping these two classifications, develop a combined classification tree, which provides meaningful terminal classes that can be used for identifying test cases. This test method can generate both valid and invalid test cases. The advantage of CTM is that, by organizing classifications and classes in the form of a tree and their hierarchical relations are made more explicit. Classification Tree Editor (CTE) [19] is used to support this CTM. It enables the tester to work interactively on the tree.

IV. COMPARISONS AND DISCUSSIONS OF IT TECHNIQUES

This section compares the IT techniques with their overview, test procedure, support tool, input, output, effectiveness, effort/cost, and types of error detected (see Table III). This table also introduces, which IT technique detects what type of error with their performance. In Table III, CT has a higher chance to detect Type 1 and Type 3 error than Type 2 or Type 4 error. For example, if the input has the correct value but it produces an incorrect output then it gives an invalid test case. This type of error occurs many times and it is easy to identify so there is a high chance for detecting Type 4 error.

Similarly, in CBC, there is a higher chance to detect Type 1, Type 3, and Type 4 errors than Type 2 error. In DFT there is a higher change to detect Type 1 and Type 2 errors than Type 3 or Type 4 error. In IM there is a higher chance to detect Type 1, Type 2 and Type 3 errors than Type 4 error. From these comparisons and discussions the testers can easily identify which technique is useful for their need to identify faults early.

V. CONCLUSION

This paper introduced a new type of IE, namely the Type 4 error. This Type 4 error can specify the location of the fault which is responsible for causing incorrect outputs. This paper proposes an overview of IT techniques in the comparisons and discussions of which technique detect what type of error. This paper is very useful to practitioners who are performing integration testing on software development. Future work includes evaluation of the relative strengths and weakness of the IT techniques with an example based on type of IE detected.

REFERENCES

[1] M. E. Delamaro, J. C. Maldonado, and A. P. Mathur, “Interface mutation: an approach for integration testing,” IEEE Transactions on Software Engineering, vol.27, no. 3, pp. 228-247, March 2001.

[2] Z. Jin, and J. Offutt. “Coupling-based criteria for integration testing,” The Journal of Software Testing, Verification, and Reliability, vol.8, no. 3, pp. 133-154, September 1998.

[3] A. J. Offutt, A. Abdurazik, and R. T. Alexander. “An analysis tool for coupling-based integration testing,” The Sixth IEEE International Conference on Engineering of Complex Computer Systems (ICECCS ’00), Japan, September 2000, pp. 172–178.

[4] W. K. Chan, T.Y. Chen, and T. H. Tse, “An overview of integration testing techniques for object-oriented programs,” Proceedings of the 2nd ACIS Annual International Conference on Computer and Information Science (ICIS), Michigan, 2002.

[5] Z. Jin, A. Offutt, “Integration testing based on software couplings,” Proceedings of the Tenth Annual Conference on Computer Assurance, USA, June 1995, pp. 13-23.

[6] A. Haley and S. Zweben, “Development and application of a white box approach to integration testing,” The Journal of Systems and Software, vol.4, pp. 309-315, 1984.

[7] R.A. DeMillo, R.J. Lipton, and F.G. Sayward, “Hints on test data selection: help for the practicing programmer,” Computer, vol.11, no.4, April 1978.

[8] A.J. Offutt, G. Rothermel, and C. Zapf, “An experimental evaluation of selective mutation,” Proceedings of the 15th International Conference on Software Engineering, May 1993, pp. 100-107.

[9] L.L. Constantine, and E. Yourdon, Structural Design. NJ: Prentice-Hall, Englewood Cliffs, 1979.

[10] S. Rapps, and E. Weyuker. “Selecting software test data using data flow information,” IEEE Transactions on Software Engineering, vol.11, no.4, pp. 367-375, April 1985.

[11] P.G. Frankl and E.J. Weyuker. “An applicable family of data flow testing criteria,” IEEE Transactions on Software Engineering, vol. 14, no.10, pp. 1483-1498, October 1988.

[12] M. Harrold, and M. Soffa, “Interprocedural data flow testing,” Proceedings of the ACM SIGSOFT '89 Third Symposium on Software Testing, Analysis, and Verification, vol.14, no.8, November 1989.

[13] E. Duesterwald, R. Gupta, and M. L. Soffa, “A demand-driven analyzer data flow testing at the integration level,” International Conference on Software Engineering, 1996, pp. 575-584.

[14] M. Grochtmann, and K. Grimm, “Classification trees for partition testing,” Software Testing, Verification & Reliability, John Wiley & Sons, Ltd, vol.3, no.2, pp. 63-82, 1993.

[15] D. J. Richardson, and L. A. Clarke, “Partition analysis: a method combining testing and verification,” IEEE Transactions on Software Engineering, vol. 11, no. 12, pp. 1477-1490, 1985.

[16] M. J. Balcer, W. Hasling, and T. Ostrand, “Automatic generation of test scripts from formal test specifications,” Proceedings of the 3rd ACM Annual Symposium on Software Testing, Analysis and Verification, ACM Press, IEEE-CS, SIGSOFT, 1989, pp. 210-218.

[17] T. J. Ostrand, and M. J, Balcer, “The category-partition method for specifying and generating functional tests,” Communications of the ACM, vol.31, no. 6, pp. 676-686, 1988.
[18] H. Leung, and P. Paramasivam, “Testing COTS with classification-tree method,” IASTED International Conference on Software Engineering and Applications (SEA), ACTA Press, L.A., U.S.A, November 2003, pp. 270-276.
[19] H. Singh, M. Conrad, and S. Sadeghipour, “Test data design based on Z and the classification-tree method,” Proceedings of First IEEE International Conference on Formal Engineering Methods, 1997.

Paramasivam Prema received her M.C.A degree from The University of Madras in the year 1999. She has more than seven years of experience in academic / research and industry. She is currently doing Ph.D in department of Computer Applications, National Institute of Technology, Tiruchirapalli. Her current research area includes Software Testing and Software Quality.

Balakrishnan Ramadoss received the M.Tech degree in Computer science and Engineering in 1995 from the Indian Institute of Technology, Delhi. The Ph.D degree in Applied Mathematics in 1983 from Indian Institute of Technology, Powai. Currently he is working as a Professor of Computer Applications at National Institute of Technology, Tiruchirapalli. His current research area includes: Software Testing Methodologies, Software Metrics, Data Warehouse – EAI, Data Mining, WBL, and XML. He is a recipient of Best Teacher Award at National Institute of Technology, Tiruchirapalli, during 2006-2007. He is a Life Member (LM) of **ISTE**, New Delhi, Life Member (LM), Computer Society of India.

Open Science Index, Computer and Information Engineering Vol:2, No:6, 2008 publications.waset.org/8525/pdf