International Journal of Software Engineering and Its Applications
Vol. 9, No. 5 (2015), pp. 61-74
http://dx.doi.org/10.14257/ijseia.2015.9.5.07

Model-Based Test-Suite Minimization using Modified Condition/Decision Coverage (MC/DC)

Ajay Kumar Jena[1], Santosh Kumar Swain[2] and Durga Prasad Mohapatra[3]

[1]*School of Computer Engineering, KIIT University, Bhubaneswar Odisha, India*
[2]*School of Computer Engineering, KIIT University, Bhubaneswar Odisha, India*
[3]*Dept. of Computer Sc. & Engg. National Institute of Technology Rourkela, Odisha, India*
[1]*ajay.bbs.in@gmail.com*, [2]*sswainfcs@kiit.ac.in*,[3]*durga@nitrkl.ac.in*

Abstract

Testing is very expensive for high-assurance software, like commercial aircraft systems, weapon research, weather forecast, earthquake forecast, and software used for safety critical system. A small and simple flaw in the end product can be enough for destroying the entire effort of the developer with a huge unrecoverable damage to the society. For this reason, Federal Aviation Administration's requirement is that, the test-suites should be comprises of Modified Condition/Decision Coverage (MC/DC) adequate. By using logic coverage criteria lots of flaws can be removed for safety critical software. MC/DC was proposed by NASA, and had been widely accepted in the field of testing. MC/DC is an effective verification technique, and helps to uncover safety faults. It is a challenge to minimize the number of test-suites when there is a partial change in the software. This can be achieved by using models. Unified Modeling Language (UML) not only helps to design software but also plays a vital role in detecting the faults early phase of design and in minimizing the test-suite. Existing test-suite minimization techniques investigated by different researchers may not be effective in minimizing MC/DC-adequate test-suites because they do not consider the complexity of the present software. A new approach for test-suite minimization is presented in this work, using dissimilarity matrix, which can be well fitted with MC/DC. We also present the results generated out of a case study of the test-suite minimization.

Keywords: *Modified Condition/Decision Coverage, test-suite minimization, dissimilarity matrix, UML*

1. Introduction

Testing is an important phase of software development which aims at producing highly reliable system while maintaining the quality. It is one of the most important and crucial activities carried out in software development process. Specially, testing is very expensive for high assurance software such as software produced for commercial aircraft system. In aerospace and safety critical domains, the quality assurance of the software is strict to the regulations of DO-178B [1]. After a small change in the existing software, new test cases are generated for the test-suite, the size of the test-suite increases, and the cost of regression testing increases. Specially in case of model-based test-suite generation, where the early detection of the flaws is possible, MC/DC also adds another feature to it for detecting the safety faults. Present day software are very complex looking to the expectation of the users. So there is a tremendous increase in the volume of test cases as well as test-suites. There may be unnecessary test cases in the test suite including both

ISSN: 1738-9984 IJSEIA

redundant and obsolete test cases. Due to the obsolete and redundant test cases, the size of the test suite continues to grow and becomes a challenge to minimize the size of the test-suite. Model-based testing is more efficient and effective than code based approach as it is the mixed approach of source code and specification requirements for testing the software [2]. Models are the intermediate artifacts between source code and requirement specification. Models preserve the essential information from requirement specification and are base for the final implementation. Unified Modelling Language (UML) is the modelling language, achieving a great attention as the industrial de-facto standard for modelling object-oriented software systems in testing. UML accomplish the visualization of software at early stage, which helps in many ways like confidence of both developer and the end user on the system, earlier error detection through proper analysis of design and etc. UML also helps in making the proper documentation of the software and so maintains the consistency in between the specification and design document. Coverage criteria are very important in test case generation. We have used MC/DC coverage criteria to generate the test cases. So, more test cases can be generated.

Regression testing ensures that, new flaws can not be generated in the unmodified part of the software when software is modified. Half of the maintenance costs is spent on regression testing whenever there is a modification in the software [3, 4]. Researchers have proposed various approaches of test-suite minimization (TSM) [2, 5], Regression Test Selection (RTS) [5, 6] and Test Case Prioritization (TCP) [7-9] techniques to minimize the cost evolved and to increase the effectiveness of regression testing. Quite a large number of test cases are generated for a very simple software. For present day complicated software a massive amount of test cases are generated. The size of the test cases increased in case of software, such as for commercial aircraft systems, weapon research, weather forecast, earthquake forecast and software evolves for highly safety critical system. It is very tedious and expensive to run all the test cases. So, it is desirable to reduce the number of test cases or to select a test-suite which is the subset of the test cases by removing some test cases from the test-suite known as test-suite minimization [1, 10]. The test-suite minimization techniques proposed by the researchers may not be effective in minimizing by the use of MC/DC adequate.

Most of the researchers have not given their proper attention minutely in using MC/DC criterion for minimization of the test-suite using model based technique which gives a motivation to us. By which we, desired to emphasize our work test-suite minimization based on the activity diagram. In this paper, we propose a frame work to minimize the test-suites. We have named our framework as MCDCTSM(MC/DC Test Suite Minimization). First of all we design the model of system in UML. After that we generate the XMI code by using XMI code generator and from that identified the predicates. Then we passed the predicates to dissimilarity matrix minimizer and thus the minimized MC/Dc test-suite are achieved.

Model-based test-suite minimization using MC/DC approach in five different case studies are reported in our approach. Our empirical study shows the encouraging results in the minimization of the size of the test-suite. By using the new approach of dissimilarity matrix, which can be well fitted with MC/DC helps us to minimize the test-suite. The dissimilarity matrix is the number of mismatches between the columns of any matrix basically in a Boolean expression.

The rest of the paper is organised as follows: Section 2, presents the basic ideas about MC/DC criterion. A brief outline of the related works is given in Section 3. Our approach is explained in Section 4 with the help of one example, to discuss how to minimize the test cases by using MC/DC. Section 5 discusses the proposed methodology for our approach. Case study and its implementation appear in Section 6. We present the empirical results of our proposed approach in Section 7. Comparison with some related works are given in Section 8. Finally, Section 9 gives the conclusion and future work.

2. Basic Ideas

In this Section, first we discussed some useful definitions which are used in our approach. Then, we discuss the basic concepts and technique for Boolean derivative methods for MC/DC coverage.

Condition : A condition is a leaf-level Boolean expression, which can not be further divided into smaller Boolean expression[11].

Group of Conditions: A Boolean statement made from a couple of conditions and one or more operators is addressed as a grouping of conditions[29].

Example: if ((A || B) && (C || D)).

In this example A, B, C and D are four different conditions and (A || B), (C || D) are two groups of conditions.

Decision or Predicate: The Boolean statement which consists of conditions and zero or many Boolean operators is called as decision or predicate.

Example: Assume an example : if((A>10) ||((B<20) &&(C>30))

In this example the if-statement and the entire expression is called a decision, || and && are the Boolean operators and (A> 10), (B < 20) and (C > 30) are conditions.

Condition Coverage: Every condition or clause in a decision in the code must take all possible outcomes at least once.

Decision Coverage: Any place of entry and exit inside program has become called at least one time, every decision inside program has taken all possible outcomes at the very least once.

Modified Condition / Decision Coverage: Any place of entry and exit inside program has been called at least one time, every decision inside the program has taken all possible outcomes at least once, and every condition in a decision has been shown to independently affect that decision's outcome.

MC/DC was designed to have Multiple Condition testing when retaining the linear growth of the test cases. The main aim of this testing is that, in the application code each and every condition in a decision statement affects the outcome of the statement [10, 11]. MC/DC needs to satisfy the followings:

- Each point of entry and exit in the code is invoked.
- Every condition in a determination statement is exercised for each and every possible output.
- Each individual possible output of any decision statement is exercised
- Every single condition in a statement is shown to independently affect the end result of the decision.

2.1. Test-suite Minimization

Changes made in the existing software may change the behaviour of the software. Test-suites tend to growing larger, often rendering it too costly to execute the entire test-suite. Test-suite minimization techniques significantly reduce how large the test-suites. But, an important issue manages how well these reduced suites can be weighed against their corresponding un-reduced suites using some other criteria rather than using the suite size criterion. Since objective of test case execution is always to detect faults on the software, one measure of the suite quality is its fault detection capability.

Test-suite minimization techniques try to remove redundant test cases of a test-suite. The test-suite minimization problem can be formally stated as follows. Given:

1. A test-suite TS of test cases {TC1, TC2, TC3,. . ., TCk}.
2. A set of testing requirements {r1, r2, r3,. . ., rn} that must be satisfied to provide the desired testing coverage of the program.
3. Subsets {T1, T2, T3,. . ., Tn} of TS, one associated with each of the r_i's, such that any one of the test cases tj belonging to T_i satisfies r_i.

2.2. Dissimilarity Matrix

The dissimilarity matrix is defined as the number of mismatches between the columns of any matrix. The mathematical model of the dissimilarity matrix can be represented by

$$D = \begin{cases} Đ & if \quad D[i,j] \; != \; D[i+1,j] \\ √ & if \quad D[i,j] = \; D[i+1,j] \\ • & Otherwise \end{cases} \qquad \text{---------- 1}$$

where i=1 to n represents the no. of rows in the matrix,
j=1 to m represents the no. of columns in the matrix

By using Equation 1 we have formulated the dissimilarity matrix shown below with the help of a Boolean expression given below in the Table 1 and Table 2. TC1, TC2, TC3 and TC4 are the test cases of the expression ((A&&B)¦¦C).

Table 1. Dissimilarity Matrix for Some Test Cases of the Expression ((A&&B)¦¦C)

Test Case#	A	B	C	!A	!B	!C	Result1 (T)	Result2 (F)
TC1	√	√	√	•	•	Đ	√	•
TC2	√	√	Đ	•	•	√	√	•

Table 2. Dissimilarity Matrix for Some Test Cases of the Expression ((A&&B)¦¦C)

Test Case#	A	B	C	!A	!B	!C	Result1 (T)	Result2 (F)
TC3	√	•	√	•	√	Đ	√	Đ
TC4	√	•	Đ	•	√	√	Đ	√

In the above example in the table we noticed that in comparison to test cases T1 and T2, the dissimilarity matrix is given as the symbol (Đ) *i.e.*, there is no existing similarities in the columns of 4 and 7. So there are 2 (two) dissimilarity in the test cases. Similarly, in comparison with test cases T3 and T4 the dissimilarity matrix is given as (Đ) *i.e.*, there is no existing similarities in the columns of 4, 7, 8 and 9. Therefore, there are 4 (four) no. of dissimilarity. A Boolean expression which has minimum number of 4 (four) dissimilarity matrix between the test case pairs will be considered as MC/DC adequate. The more the number of dissimilarity between the test cases then more suitable for MC/DC.

3. Related Works

A good number of research work have been on the size of reducing the test-suites. But very few work / no work have been done for test-suite minimization using UML design and MC/DC criterion.

Jones *et al.*, [11] presented an algorithm for test-suite minimization and prioritization by using MC/DC criterion. They proposed an algorithm to reduce the size of the test-suite and to prioritize the test cases. The empirical studies have shown that the size of the test-suite has been reduced significantly. The proposed test-suite minimization algorithm, utilizes an extra approach to recomputed the contribution of test cases after each test case is selected and uses a break down approach that removes the weakest test cases from the test-suite.

Blue *et al.*, [13] proposed an approach Interaction-based test-suite minimization (ITSM) as a complementary approach to standard combinatorial test design. They have considered two real life case studies where standard combinatorial test design was applicable and ITSM was used successfully. They also discussed different approaches to solve ITSM problem. Finally, they analysed the experimental results.

Harrold *et al.*, [14] have presented a technique which manages the test-suite by identifying the redundant test cases. They conducted three separate experiments of varying sizes to determine the possible minimization of the test-suite size making using their techniques. Wong *et al.*, [15] performed a study on the effect of test set minimization on fault detection effectiveness by using the tool ATACMIN [16].

Almeida *et al.*, [17] presented an approach to generate test cases automatically using MC/DC criteria from requirements represented in XML. They also converted the XML format to a graph to represent system model and transitions by using its states. They developed a Java prototype to implement their approach. After the system graph they processed the condition of each requirement to generate the test cases.

Godboley *et al.*, [12] proposed an automated procedure for the generation of test-suite which helps in achieving and increase in MC/DC coverage of a program. By using code transformation technique they identify the predicates and the empty if-else statements based on exclusive nor (X-NOR) operations. They use a tool CREST TOOL to generate the test suite. But they did not address for a group of programs or for integrated programs.

4. Test-suite Minimization Example

Let us consider the Boolean expression ((A&&B)||(C&&D)). The possible test cases with MC/DC pairs are given in Table 3. For calculating the MC/DC table the reader may follow [1].

Table 3. The Independence of Conditions in Test Cases using MC/DC

Test Case ID	A	B	C	D		Z		A	B	C	D
1	F	F	F	F		F					
2	F	F	F	T		F				4	
3	F	F	T	F		F					4
4	F	F	T	T		T				2	3
5	F	T	F	F		F					
6	F	T	F	T		F					
7	F	T	T	F		F					8
8	F	T	T	T		T					7
9	T	F	F	F		F			13		
10	T	F	F	T		F				12	

11	T	F	T	F		F			15		12
12	T	F	T	T		T				10	11
13	T	T	F	F		T			9		
14	T	T	F	T		T					
15	T	T	T	F		T			11		
16	T	T	T	T		T					

The independence of condition B, the test cases are 9+11+13+15. The independence of condition C, the test cases are 2+4+10+12. The independence of condition D, the test cases are 3+4+7+8+11+12. The resulting test cases are 2,3,4,7,8,9,10,11,12,13,15 *i.e.*, FFFT, FFTF, FFTT, FTTF, FTTT, TFFF,TFFT, TFTF, TFTT, TTFF, and TTTF.

Table 4 represents the dissimilar matrix of the expression ((A&&B) ¦¦(C&&D)). The same is implemented in Java. Table 5 shows the number of dissimilarity between all pairs of the test cases *i.e.*, the first row is given as between T1 to (T1, T2, T3,..... ,T16). Similarly the second row as T2 to (T1, T2, T3,...., T16) and so on. Then we calculated the sum of 4 (four) in each column and presented in the last row of the table.

Table 4. Dissimilarity Matrix of the Boolean Expression ((A&&B) ¦¦(C&&D))

Test Case#	A	B	C	D	!A	!B	!C	!D	Result1 TRUE	Result2 FALSE
1					√	√	√	√		√
2				√	√	√	√			√
3			√		√	√		√		√
4			√	√	√	√			√	
5		√			√		√	√		√
6		√		√	√		√			√
7		√	√		√			√		√
8		√	√	√	√				√	
9	√					√	√	√		√
10	√			√		√	√			√
11	√		√			√		√		√
12	√		√	√		√			√	
13	√	√					√	√	√	
14	√	√		√			√		√	
15	√	√	√					√	√	
16	√	√	√	√					√	

Table 5. Dissimilarity Matrix of ((A&&B) ¦¦ (C&&D)) with other Test Cases

Test Case#	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	∑
1	0	2	2	6	2	4	4	8	2	4	4	8	6	8	8	10	4
2	2	0	4	4	4	2	6	6	4	2	6	6	8	6	10	8	4
3	2	4	0	4	4	6	2	6	4	6	2	6	8	10	6	8	4
4	6	4	4	0	6	6	6	2	8	6	6	2	8	6	6	4	3
5	2	4	4	6	0	2	2	6	4	6	6	10	4	6	6	8	4
6	4	2	6	6	2	0	4	4	6	4	8	8	6	4	8	6	5
7	4	6	2	6	2	4	0	4	6	8	4	8	6	8	4	6	6
8	8	6	6	2	6	4	4	0	10	8	8	4	6	4	4	2	5
9	2	4	4	8	4	6	6	10	0	2	2	6	4	6	6	8	4
10	4	2	6	6	6	4	8	8	2	0	4	4	6	4	8	6	6
11	4	6	2	6	6	8	4	8	2	4	0	4	4	8	4	6	6
12	8	6	6	2	10	8	8	4	6	4	4	0	6	4	4	2	5
13	6	8	8	8	4	6	6	6	4	6	4	6	0	2	2	4	3
14	8	6	10	6	6	4	8	4	6	4	8	4	2	0	4	2	5

15	8	10	6	6	6	8	4	4	6	8	4	4	2	4	0	2	5
16	10	8	8	4	8	6	6	2	8	6	6	2	4	2	2	0	2
∑	4	4	4	3	4	5	6	5	4	6	6	5	3	5	5	2	

Here we consider the test cases where the sum of the dissimilarities between the test case pairs equal to 4. Hence, we calculated the total number of 4's in the dissimilarity matrix in both row and column wise. Based on the dissimilarity matrix the final minimized test cases selected for the test-suite are obtained as follows

TSmin = {T1,T2,T3,T5,T6,T7,T8,T9,T10,T11,T12,T14,T15}

TSoriginal = {T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15}

5. Proposed Methodology

We propose an approach to generate test cases using UML activity diagram, in which a large number of decisions, conditions and predicates are available, which are required for creating the dissimilarity matrix.

5.1. Proposed Frame work: MCDCTSM

We propose a framework named MCDCTSM (MC/DC Test-Suite Minimization) whose system model is given in Figure 1. Below, we discuss the important components of our framework.

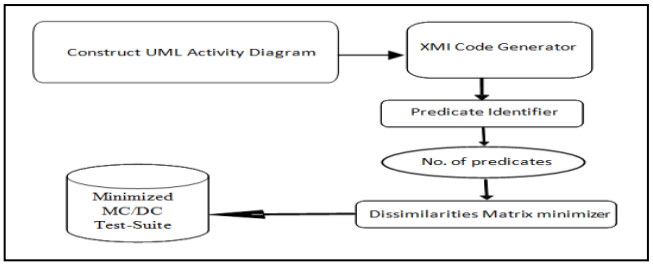

Figure 1. System Model of MCDCTSM

5.2. Create UML Diagram and Identifying the Predicates:

- First, we construct the UML activity diagram designed for the system under consideration.
- After the construction, we generate the XMI (XML Meta Interface) code considering to this activity diagram.
- Then, we write a Java program to which the XMI code is passed and it will identify the number of predicates as well as the predicates and stores it in another (.txt or .doc) document or text file. The predicate identifier will identify the predicates by using Algorithm 1.
- Finally, the dissimilarity matrix minimizer is a Java program which will take the input of a .txt file in which the number of predicates and predicates are stored and processed to find the Minimized MC/DC test-suite by using Algorithm 2. The original and minimised test-suites are given in Figure 5. As in this heading, they should.

6. Case Study and its Implementation

For the generation of test cases, we have used ATM withdrawal system as the case study using some graphical method [18, 19]. We have also used some other

case studies like book issue use case of library information system, on line shopping system, coffee vending machine, user login screen and air ticket reservation system [18]. The case studies are prepared by using different models of UML. Mainly the activity, sequence, state chart and interaction overview diagrams are considered for design of case studies. The snapshots of the activity diagram and its corresponding XMI file for the ATM withdraw system are presented in Figures 2 and 3 respectively. The XMI file is passed to a predicate analyser written in Java and captured the no of predicates written into a data file. The data file is again passed to the dissimilarity matrix minimiser and the minimised MC/DC test-suite is generated.

Algorithm 1 Test-suite Minimizer
Input : UML Diagram(X)
Output: Minimized MC/DC Test-suite-list

1. *No_of_predicate ← 0*
2. *X ←Activity Diagram prepared in RSA Tool for the system under test*
3. *XMI_Code [ ] ← Φ*
4. *List_Predicate [ ] ← Φ*
5. *XMI_Code [ ] ← gen_of_XMI(X)*
6. *For each statement S ∈ XMI_Code*
7. *If(&& or || or ! in S)*
8. *strcat(List_Predicate,S)*
9. *No_of_predicate ← No_of_predicate + 1*
10. *EndIf*
11. *EndFor*
12. *Minimized_MC_DC test-suite-list ←DMM(No_of_predicate)*
13. *Exit*

Algorithm 1 is to calculate the number of predicates and to produce an array which stores the predicates available in the XMI code. First of all it will accept the UML activity model of the system under test. Then the corresponding XMI code will be stored in line number 6 to XMI_code. Then for each statement in the XMI_code it will calculate the predicates and stores it in No_of_Predicates. If it will find any && or || or ! in the statement then it will append the list List_Predicate in line 8. Then No_of_Predicates will be passed to DMM to find the minimized test-suite.

Algorithm 2 DMM
Input : No_of_predicate
Output: Minimized MC/DC Test-suite-list

1. *A[ ] ← Initialize values of n predicates to the array*
2. *TSL [ ] ← Φ*
3. *n ← No_of_predicate*
4. *for i ← 0 to 2^n*
5. *n_4 ←0*
6. *for j ← 0 to 2^n*
7. *de ← 0*
8. *for k ← 0 to 2n+2*
9. *If(A_{ik} != A_{jk})*
10. *de ← de+1*
11. *End if*
12. *End for*
13. *If (de==4)*
14. *n_4 ← n_4 + 1*
15. *End if*
16. *End for*
17. *A_{i2n} ← n_4*
18. *j ← 0*
19. *for i ← 0 to 2^n*
20. *If(A_{i2n}= 4)*
21. *TSL[j++] ← A_{i2n}*
22. *EndIf*
23. *EndFor*
24. *End for*
25. *Minimized_MC_DC test_suite_list ←TSL*
26. *Exit*

Algorithm 2 is a function call from Algorithm 1 which initializes the values of the predicates in the array A and initializes test-suite list (TSL) to NULL. By the concept used in Section 2.2 it will calculate the minimized test-suite list.

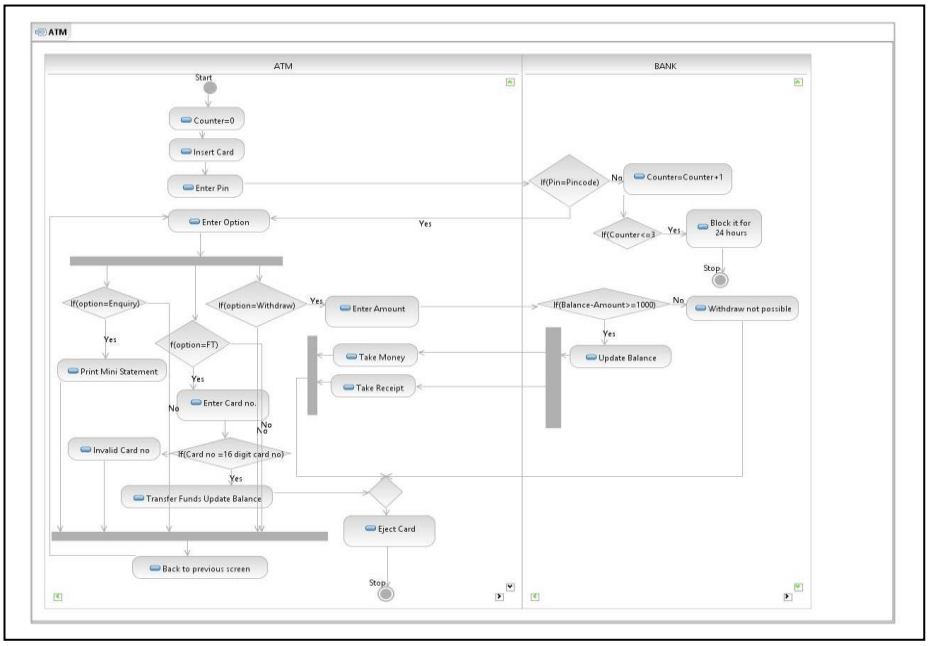

Figure 2. Activity Diagram of ATM Withdrawal System

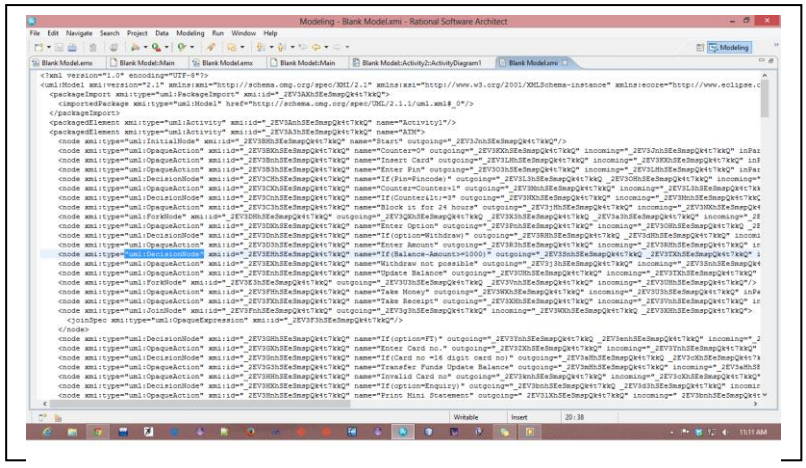

Figure 3. Snapshot of the Generated XMI Code of Activity Diagram given in Figure 2

7. Empirical Results

In this section, we present the results obtained after the empirical study done to search the effectiveness of our model-based test-suite minimization technique. In our approach, we consider a case study of ATM system as the case study. The detailed is presented in section 5. We also implement it in other four different case studies. The snap shot of the predicate analyser is given in Figure 4. From the dissimilarity matrix, the original and minimized test-suites are prepared. Table 6 shows the original test-suite ($TS_{original)}$ and minimized test-suites ($TS_{min)}$ for each case study. Figure 6 represents a graphical correlation between original test-suite vs minimized test-suite.

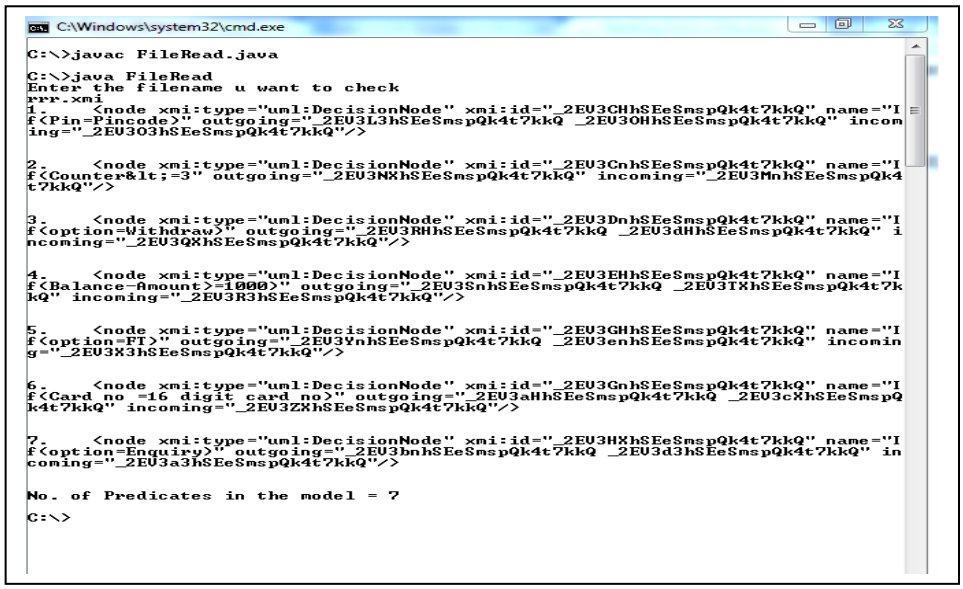

Figure 4. Snapshot Showing the Number of Total Predicates Identified in the XMI

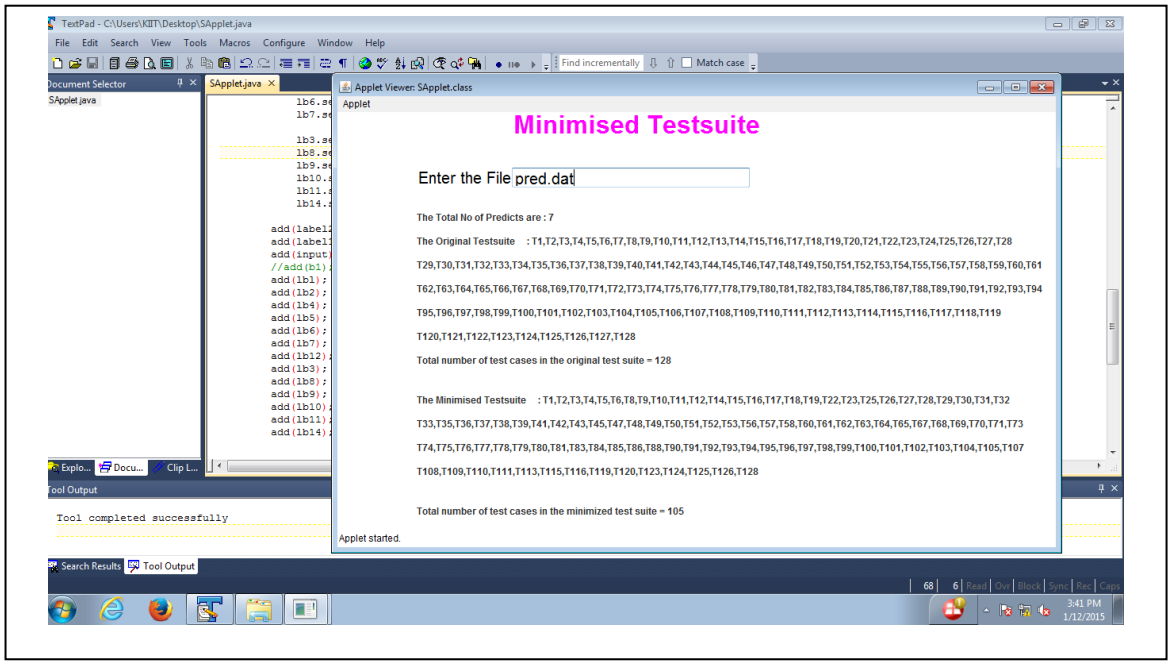

Figure 5. Snapshot showing the Original Test-suite and the Minimised Test-Suite

Table 6. Test-suite Minimization Results for Five Real Life Problems

Sl. No.	Name of the case study	Number of test cases in the original test suite	Number of test cases in the minimized test suite	% of Reduction
1	ATM Withdrawal System (ATMWS)	25	19	82.0
2	Book Issue of Library Information System	63	51	80.5

	(BILIS)			
3	User Login System (ULS)	24	19	78.1
4	Online Shopping System (OSS)	86	74	82.5
5	Online Ticket Reservation System (OTRS)	47	35	84.7

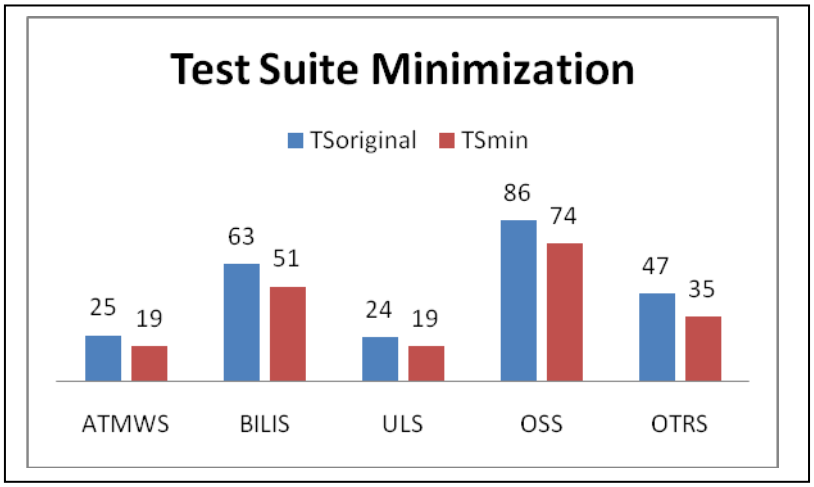

Figure 6. Correlation between Original Test-Suite vs. Minimized Test-Suite your Main Text in 11-point

8. Comparison with Related Works

For In this Section, we compare our work with some existing work using UML models.

Model based test case minimization has been discussed with a large scope in the literature [13]. But, to the best of our knowledge no work/ very few works have been reported till date using MC/DC coverage and UML models for the minimization of the test-suites.

Jones *et al.*, [11] presented an algorithm for test-suite reduction and prioritization by using MC/DC criterion. The empirical studies have shown that the size of the test-suite has been reduced significantly. The proposed test-suite minimization algorithm, utilizes an extra approach to re-compute the contribution of test cases after each test case is selected and uses a break down approach that removes the weakest test cases from the test-suite. They have not used any UML diagrams for the minimization. In comparison to their work, we generate the test cases in the earlier stages of the design process. So that the flaws can be identified and eradicated in early stage. Though we uses MC/DC coverage for minimization, a small change in the predicates may affect the results which helps us to discover more flaws in the beginning phase.

Blue *et al.*, [13] proposed an approach Interaction-based test-suite minimization (ITSM) as a complementary approach to standard combinatorial test design. They have considered two real life case studies where standard combinatorial test design was applicable and ITSM was used successfully. They also discussed different approaches to solve ITSM problem. In comparison to this approach we presented a model and then tried to minimize the test-suite. Our approach is fully automated where they [13] proposed a semi automatic approach.

Our approach is also comparable to the approach of [17] in which the authors generated the test cases based on MC/DC criteria from XML. For the generation of test

cases, they represent system model and transitions by using its states. In our approach we use UML models. In our approach interaction faults will be identified from the design model.

9. Conclusion and Future Work

MC/DC is a high-level coverage criterion based on control flow analysis. Though there are plenty test-suite minimization issues addressed in the literature, most of them consider the statement and branch coverage criterion. We proposed a framework named MCDCTSM for test-suite minimization by using dissimilarity matrix. We discussed the details of our approach along with a case study. In our approach we considered the MC/DC coverage criterion and noticed that the test-suite size has significantly minimized. We have considered five different case studies and applied our proposed algorithms and noticed the improvement in minimization of test-suite. The main threats of validity our proposed approach is that, it is only confined to the activity, sequence, interaction overview and state chart diagram of UML models. The advantage of this approach is to achieve a tremendous increase in MC/DC coverage in model-based testing. It encouraged us to have further investigation on measuring the fault detection in comparison to other proposed algorithms and to prioritize the test-suite using MC/DC. Our future work also includes minimizing the test-suite by using Karnaugh Map.

References

[1] K. J. Hayhurst, S. D. Veerhusen, J. J. Chilenski and K.L. Rierson, "A Practical Tutorial on Modified Condition/Decision Coverage", NASA / TM-2001-210876, **(2001)**.
[2] B. Korel and L. Tahat, "Model-based regression test minimization using dependence analysis", International Conference on Software Maintenance, **(2002)**, pp. 214-223.
[3] G. Kapfhammer, "The Computer Science Handbook", Chapter Software testing. CRC Press, Boca Raton, FL, (**2004**)
[4] H. Leung and L. White, "Insights into regression testing", Proceedings of the Conference on Software Maintenance, **(1989)**, pp. 60-69.
[5] M. Harrold, J. Jones, T. Li, D. Liang and A. Orso, "Regression test selection for java software", Proceedings of the 16th ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages and Applications, **(2001)**, pp. 312-326.
[6] G. Rothermel, M. Harrold and J. Dedhia, "Regression test selection for C++ software", SoftwareTesting, Verification and Reliability, **(2000)**, pp. 77-109.
[7] S. Elbaum, A. G. Malishevsky and G. Rothermel, "Test case prioritization: A family of empirical studies", IEEE Transactions on Software Engineering, vol. 28, no. 2, **(2002)**, pp. 159-182.
[8] A. Malishevsky, J. Ruthruff, G. Rothermel and S. Elbaum, "Cost-cognizant test case prioritization", Technical Report TR-UNL-CSE-2006-0004, **(2006)**.
[9] C. Panigrahi and R. Mall, "Model-based test case prioritization", ACM SIGSOFT Software Engineering Notes, vol. 35, no. 6, **(2010)**, pp. 1-7.
[10] Z. Awedikian, K. Ayari, and G. Antoniol, "MC/DC automatic test input data generation", Proceedings of the 11th Annual conference on Genetic and evolutionary computation, GECCO '09, (New York, USA), ACM, **(2009)**, pp. 1657-1664.
[11] J. A. Jones and M. Jean Harrold, "Test-suite reduction and prioritization for MC/DC", IEEE Transactions on Software Engineering, vol. 29, no. 3, **(2003)** March, pp. 195-209.
[12] S. Godboley, G. S. Prashanth, D. P. Mohatra and B. Majhi, "Enhanced Modified Condition / Decision Coverage using Exclusive-Nor Code Transformer", IEEE International Conference, **(2013)**, pp. 524-531.
[13] D. Blue, I. Segall, R. Tzoref-Brill and A. Zlotnick, "Interaction-Based Test-suite minimization", IEEE International Conference, ICSE, San Francisco, CA, **(2013)**, pp. 182-191.
[14] M. Jean Harrold, Rajiv Gupta and Mary Lou Soffa, "A methodology for controlling the size of a test-suite", ACM Transactions on Software Engineering and Methodology, vol. 2, no. 3, **(1993)**, pp. 270-285.
[15] W. E. Wong, J. R. Horgan, S. London and A. P. Mathur, "Effect of test set minimization on fault detection effectiveness", Software – Practice and Experience, vol. 28, no. 4, **(1998)**, pp. 347-369.
[16] J. R. Horgan and S. A. London, "ATAC: A data flow coverage testing tool for C", Proceedings of Symposium on Assessment of Quality Sofiware Development Tools, New Orleans, LA, **(1992)**, pp. 2-10.
[17] M. A. Almeida, J. M. Bezerra and C. M. Hirata, "Automatic generation of test case for critical systems based on MC/DC criteria", Digital Avionics Systems Conference (DASC), IEEE/AIAA 32nd, **(2013)**, pp. 7C5-1-7C5-10.

[18] A. K. Jena, S.K. Swain and D. P. Mohapatra, "A novel approach for test case generation using activity diagram", IEEE International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT), 2014, New Delhi, **(2014)**, pp. 626–634.
[19] H. Das, A. K. Jena, P. K. Rath, B. Muduli and S. R. Das, "Grid computing based performance analysis of power system: A graph theoretic approach", Proceedings of the Intelligent Computing, communication and devices, Springer, **(2014)**, pp. 259-266.

Authors

Ajay Kumar Jena received his M. Tech. degree in Computer Science and Engineering from BPUT University, Odisha, India in 2009. He has over 6 years of teaching experience. He is currently working as Full Time Research Scholar in School of Computer Engineering, KIIT University, Bhubaneswar, India. He has published one paper in a journal and several papers in conferences. His current research interests include object-oriented software testing, model-based testing, and complex network analysis.

Santosh Kumar Swain is presently working as Associate Professor, School of Computer Engineering, KIIT University, Bhubaneswar, Odisha India. He has acquired his M.Tech degree from Utkal University, Bhubaneswar and Ph. D. from KIIT, University, Bhubaneswar, India.. He has contributed more than 25 papers to Journals and Proceedings. He has written one book on "Fundamentals of Computer and Programming in C". His interests are in Software Engineering, Object Oriented Systems, Sensor Network and Compiler Design etc.

Durga Prasad Mohapatra did his M. E. at National Institute of Technology, Rourkela, India. He has received his Ph. D. in Computer Science & Engineering from Indian Institute of Technology Kharagpur, India. He has over 19 years of teaching experience. His research interests include Software Engineering, Program Slicing, Software Testing, Distributed System and Real Time System. He has more than 45 number of research publications in his credit. He is the author of the book "Elements of Discrete Mathematics". Presently he is working as Associate Professor, Department of Computer Science and Engineering, National Institute of Technology, Rourkela, Odisha, India.