EXTENSION OF E(Θ) METRIC FOR EVALUATION OF RELIABILITY

By

SUBHAJIT MONDAL

B.E., Visveswaraiah Technological University, India, 2003

A REPORT

Submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

DEPARTMENT OF COMPUTING AND INFORMATION SCIENCES
COLLEGE OF ENGINEERING

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2005

Approved by:

Major Professor
Dr. David A. Gustafson

ABSTRACT

The calculation of reliability based on running test cases refers to the probability of the software *not* generating faulty output consequent to the testing process. The metric used to measure this reliability is referred in terms of E(Θ) value. The concept of E(Θ) gives precise formulae to calculate the probability of failure of software after testing, debug or operational. This report aims at extending the functionalities of E(Θ) into the realm of multiple faults spread across multiple sub-domains. This generalization involves introduction of a new set of formulae for E(Θ) calculation which can account for faults spread over both single as well as multiple sub-domains in a code. The validity of the formulae is verified by matching the obtained theoretical results against the empirical data generated from running a test case simulator. The report further examines the possibility of an upper bound calculation on the derived formulae and its possible ramifications.

Index Terms – Reliability, software testing, debugging.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

At the root of any software project lays an inspiration to better things and an idea as to how to do it. Simplification of existing techniques with an eye on generalization has been the driving force behind this project. I thank my advisor Dr. David A. Gustafson for providing me with both the vision and the inspiration to take on this project. His meticulous guidance and patience in resolving issues have ensured a timely and successful completion of the project. I would also like to acknowledge my committee members, Dr. Daniel Andresen and Dr. Gurdip Singh for their valuable input and support during the project.

Since any professional achievement requires support in personal life as well; it would be ungrateful on my part not to thank the people who have ensured that my total focus be on the project during these past months. Foremost, I would like to thank my family specially my brother Sujit for always being there for me and providing the level of support which saw me through some rough patches during the project. I extend my gratitude to my friends who have made themselves readily available, though at times unwillingly, to many of my babble sessions regarding issues pertaining to the project. Sharing the frustrations of getting stuck and the exhilarating feeling of overcoming obstacles has been a roller coaster fun ride.

Last but not in a least way I would like to thank the staff of the Computing and Information Sciences department for showing me the way through the maze of administrative work involved prior to graduation.

CHAPTER 1
INTRODUCTION

The goals of software testing can be divided into two parts. One part deals with proving the correctness of the software. Another part deals with probing the software for possible faults. The interpretation of the terms “failure set” is open to arbitration but in this report we have assumed it to be an input or a sequence of input which result in an unexpected output or “failure”. By unexpected we mean a deviation from the norm.

Testing methodologies [1] include two distinct systematic paths of testing. One methodology assumes the presence of hypothetical faults in the software and aims at generating test cases targeting such faults. One of such approaches is ‘boundary testing’. This methodology targeting test cases with the intention of catching faults is ‘debug testing’. Another methodology looks at the total set of inputs expected in operation and selects a subset from that, operational testing [1]. Careful subset selection reduces the cumbersome and almost impossible process of testing every input in operational profile without affecting the outcome of the test.

The choice between the two depends on the goal target after testing. Debug testing is better suited for achieving a desired level of reliability for the software. Operational testing on the other hand facilitates reliability assessment. While the effectiveness of debug testing is highly dependent on the intuition behind the testing process, operational depends only on the selection criteria of a subset of inputs from the set of operational inputs.

Both of the approaches aim at increasing the reliability of the software after testing. This measure of reliability is known as $E(\Theta)$. It refers to the probability of the software encountering a failure subsequent to testing. A higher $E(\Theta)$ value means a lower reliability. The effectiveness of debug testing increases upon dividing the input domain into effective sub-domains. This occurs when the failure cases are concentrated in a sub-domain. The probability of uncovering a point in the failure set becomes greater with

respect to that when compared without sub-domains. We have considered a particular area of debug testing using sub-domains, multiple faults in multiple sub-domains and then generalized the method to include all possible cases.

CHAPTER 2

EMPIRICAL CALCULATION of E(Θ)

The simulator used in the experiment to generate and analyze data from empirical testing of software is based upon the probability distribution of the points in the failure set. The value of E(Θ) was calculated from the probability distribution of the faults cases occurring during testing. It is assumed that once a particular failure occurs, the fault causing that failure is discovered and thus removed. Another vital assumption is that the process of removal of faults does not introduce any new faults. Thus a software having 'x' faults when subjected to testing will have the number of faults either at the same level (x) or lower (x-i), consequent to testing such that $0< i < x$.

2.1 FAULT DISTRIBUTION

By definition E(Θ) refers to the probability of the software encountering a failure subsequent to testing. Another interpretation of the above said definition is, the probability of a fault remaining undetected in the software after testing. Let us assume a hypothetical marking scheme where we can mark the faults present in the software either 1 or 0. A value of 1 represents that the faults is not detected and a value of 0 represents that the fault is still present in the software. This means that for "y" faults in a software we have y^2 possible combination of faults existing or not existing (have been found and removed).

No.	a	b	Fault case
1.	0	0	00
2.	0	1	01
3.	1	0	10
4.	1	1	11

Table 2.1a – Possible Fault cases

No.	Fault Case	Faults Detected	Faults Remaining
1.	00	none	a, b
2.	01	b	a
3.	10	a	b
4.	11	a, b	none

Table 2.1b – Failure Regions Detected

As an example let us consider a case with 2 faults, “a” and “b”. Possible fault cases are depicted above in Table 2.1 a.

E(Θ) calculation of this particular case would involve calculating the reliability of the given software before testing minus the probability that one or more of these four fault cases occur during testing. The occurrence of any of these cases would determine which faults get detected and which still remain undetected. Figure 2.1b examines the effect of occurrence of these fault cases on the faults.

CHAPTER 3
SIMULATOR

The software used in the simulator makes the following simplifying assumptions:

a) Faults: Numbers or Points in the input domain which generate failures.
b) Input Domain: A Finite set of numbers.
c) Test Cases: Numbers which are selected from the Input domain and can potentially cause a failure.

Selection criteria:

a) Random Selection: any number from the *entire* input domain is selected randomly. This selection process is used when the input domain is not broken into sub-domains.
b) Sub-domain Selection: selection of numbers is restricted to individual input *sub-domains.*

The simulator uses the java random number generator function for selection of test cases.

3.1 VALIDATION OF THE SIMULATOR DESIGN

To ensure the correctness of the experiment, it's important to establish the accuracy of the simulator design through verifiable data. Validity of the simulator is established by comparing empirical data got from simulation with the theoretical $E(\Theta)$ values obtained from applying the formulae described in paper[1]. Due to limitation of space we would concentrate the validation proof to only a few areas of sub-domain testing.

Single Domain - Multiple failures:

Domain nature:

Single Domain = 1 – 100 (100 points)
Failure Region A = 21 – 40 (20 points)
Failure Region B = 61 – 80 (20 points)

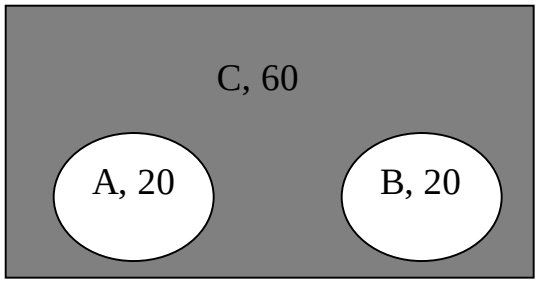


Figure 3.1 Single Domain, Multiple Failure Regions

Result Sheet:

Fault Case	Fault probability (q)	Fault frequency (p)	p*q
0	0.4	0.375	0.15
1	0.39	0.283	0.1104
2	0.2	0.259	0.0518
3	0.0	0.083	0.0

Table 3.1 – E (Θ) Result Sheet

E (Θ) = $\sum$ (p*q) = 0.31217003

It is assumed that the theoretically generated data is correct and that if the empirical data generated by the simulator matches this data then it validates the accurateness of the design. The reason behind assuming the correctness of the theoretical data lies with the research work done by Phyllis G. Frankl, Richard G. Hamlet, Bev Littlewood, and Lorenzo Strigini , in the paper, “Evaluating Testing Methods by Delivered Reliability,” published in IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 24, NO. 8, AUGUST 1998 [1].

Theoretical value of E(Θ) calculated as per formulae described in [1] is as follows:

Multiple failure regions w/o sub-domains:

$$E(\Theta) = \sum_{i=1}^{m} q_i (1 - d_i)^T$$

Where:

q_i = failure probability in that sub-domain = 20/100,

d_i = detection rate in that sub-domain = 20/100,

T = number of test cases per sub-domain = 1

Hence E(Θ) = 20/100(1- 20/100) + 20/100(1- 20/100) = 32/100 = 0.32 (quite close to the generated value)

The above comparison shows that the simulator's output matches the theoretical predictions of E(Θ) values. Since verification of the correctness of the simulator is quite critical to this experiment, the above test is but one of a series of tests which were done to validate the simulator. Another test of the simulator's validity would be its capability to track changes in E(Θ) values as the fault scenarios change, as discussed below.

3.2 SIMULATOR WORKING ENVIRONMENT

Among the many criteria taken into consideration while designing the simulator, one was ease of use. The following few snap shots show the various forms and their characteristics, in the user interface model.

Parameter Input frame:

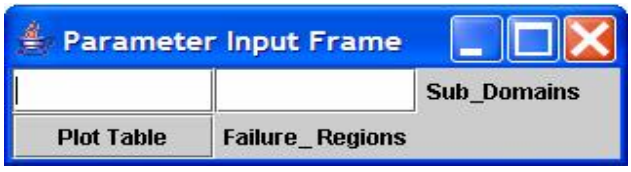


Figure 3.2a Input Frame One

Functionality:

1> Accepts the number of sub-domains and the number of failure regions

2> Plots the *table* to accept individual failure points of each sub-domain, shown below.

The "Plot Table" button upon activation creates a table capable of accepting the number of failure points in each sub-domain. The layouts of the tables thus created depend on the number of sub-domains and failure regions. We have made certain simplifying assumption:

a> Total number of sub-domains equal the total number of failure regions i.e. number of failure regions per sub-domain is never more than the total number of sub-domains.

b> Maximum number of sub-domains is three.

Sub-Domain and Failure Region Input Frame:

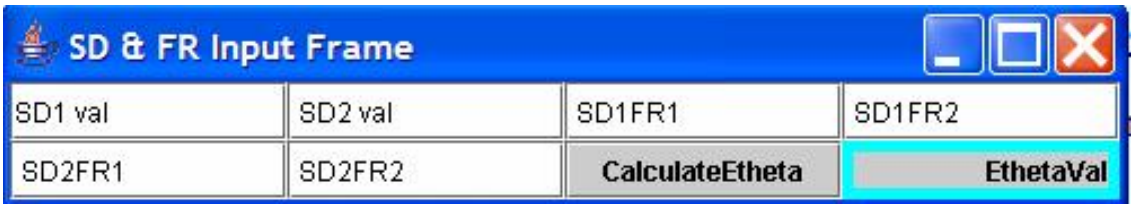


Figure 3.2b Input Frame Two

Functionality:

1> Accepts the number of input points per sub-domain

2> Accepts the number of failure points per sub-domain

3> Calculates the E(Θ) value based on the input fields.

The "CalculateEtheta" button upon activation calculates the E(Θ) value for the depicted scenario and outputs the result in the "EthetaVal" output box.

The following series of figures represent a pictorial demonstration of the process of calculation of E(Θ) for this scenario:

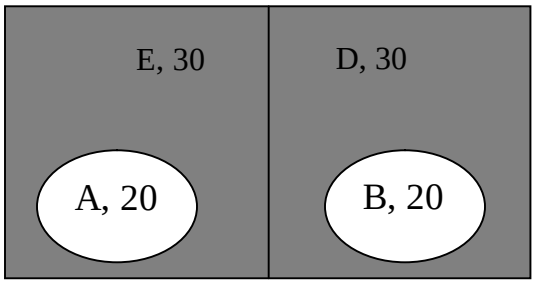


Figure 3.2c Multiple Domains Multiple Failure Regions

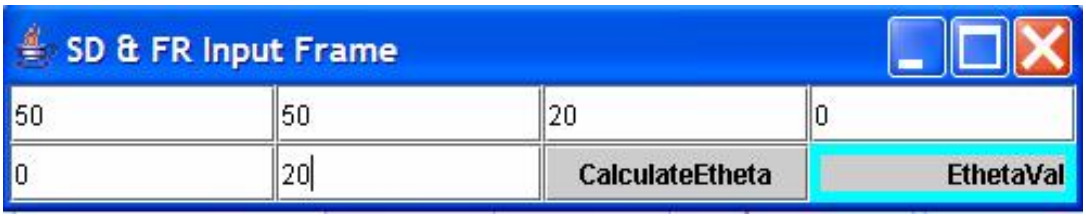


Figure 3.2d Input Frame Two with Values

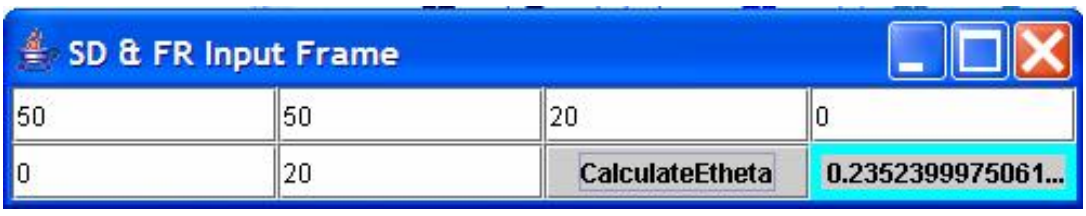


Figure 3.2e Output Frame with the Calculated E(Θ) Value

The following figure depicts the simulator input table with 3 sub-domains and 3 failure regions:

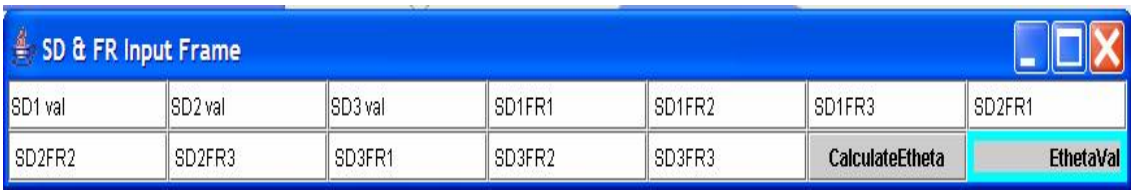


Figure 3.2f Input Frame Three

CHAPTER 4
OVERLAPPING VS. NON-OVERLAPPING FAILURE REGIONS

If a piece of software has more than one failure regions then there are two possible ways these different regions can interact. One way is that they interact independent of each other i.e. the changes in one region do not affect the properties of the other regions. This form of interaction is seen when the failure regions are non-overlapping. In non-overlapping failure regions the discovery of one region is independent of the discovery of other failure regions. The other way the regions can interact is by having properties which are dependent on each other i.e. the changes in one region affect the properties of the other regions too. This form of interaction is seen when the failure regions are overlapping. In such kind of interaction the discovery of a region may lead to the discovery of other dependent failure regions. The probability of such dependent discoveries occurring is directly proportional to the degree of overlap between the various failure regions.

It is not difficult to see why the *affect* of testing is more on software with overlapping faults in than in one with non-overlapping faults. By *affect* we mean the change in the values of $E(\Theta)$ before and after testing. The reason being that with overlapping faults a single test case can possibly discover one or more faults, whereas in non-overlapping faults the maximum number of faults one test case can discover is 1. The discovery of a fault due a failure caused by a test case leads to the fault being removed from the software. It can be shown that given similar reliability values for two software codes one with overlapping failure region and the other with non-overlapping ones, the $E(\Theta)$ values of overlapping software would always be less than that of non-overlapping ones. As mentioned in the previous section, the ability of the simulator to test the above discussed difference in $E(\Theta)$ values between software with overlapping and non-overlapping failure regions is another measure of its accuracy.

Reconfiguring the above discussed software to accommodate comparison between overlapping and non-overlapping faults.

Non- Overlapping Failure Regions

Domain Nature:

Sub-Domain 1 = 1 - 50 (50 points)

Sub-Domain 2 = 1 - 50 (50 points)

Failure Region A = 21 – 40 (20 points)

Failure Region B = 61 – 80 (20 points

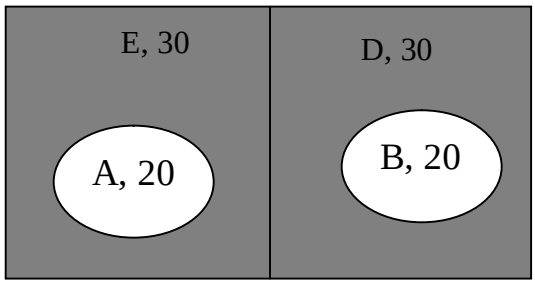


Figure 4.1 Non-Overlapping Failure Regions

E (Θ) = 0.2369

Overlapping Failure Regions

Domain Nature:

Sub-Domain 1 = 1 - 50 (50 points)

Sub-Domain 2 = 1 - 50 (50 points)

Failure Region A = 30– 45 (15 points)

Failure Region B = 30– 45 (15 points)

Failure Region C = 45 – 55 (10 points)

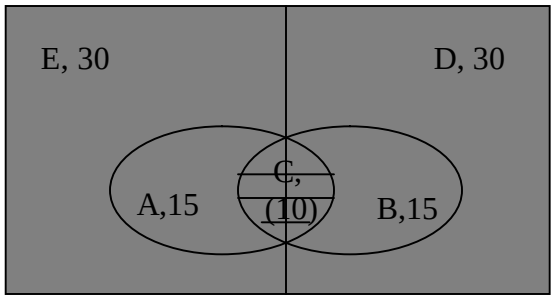


Figure 4.2 Overlapping Failure Regions

E (Θ) = 0.2013

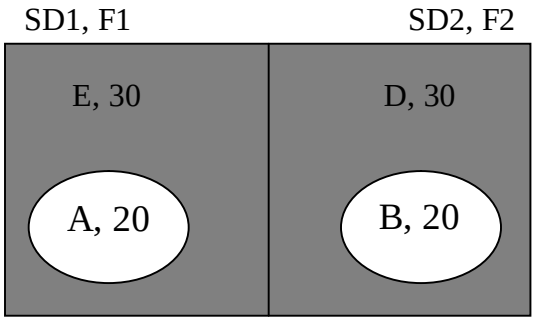
It can be seen that in the two situations discussed above the sizes of the total input domain, sub-domain size and the sizes of the failure sets were identical. The only difference among these was the degree of interaction between the two failure regions. One had non-overlapping or independent failure regions while the other had overlapping or dependent failure regions. Hence the difference in the E(Θ) values can be attributed to this single difference. As Expected the simulator results show that E(Θ) (Overlapping) > E(Θ) (Non-Overlapping).

CHAPTER 5

MULTIPLE FAILURE REGIONS IN MULTIPLE SUB-DOMAINS

Now that the accuracy of the simulator design has been verified, we take our experiment to the next level which is to calculate E(Θ) values theoretically for domains with multiple sub-domains. Simulation of testing on software with multiple failures spread across multiple sub-domains gives an insight to formulating a method to calculate E(Θ) non-empirically. We ran the simulations on software with multiple failure regions spread across multiple sub-domains. The method used by the simulator and the data thus got are interpreted into theoretical models as explained in the following section.

Going back to the model described in *Fig. 4.1*:



	F1	F2
SD1	20/50	0/50
SD2	0/50	20/50

Figure 5.1 Non-Overlapping Failure Regions (NOFR) *Table 5.1 – NOFR*

There is one test-case per sub-domain. Test-1 value in table is the region the test-case discovered. Failure regions are region A in sub-domain 1 and region B in sub-domain 2. Regions D and E are non-failure regions.

Test-1	Test-2	F1	F2	Case number
D	E	0	0	Seq1
D	B	0	1	Seq2
A	E	1	0	Seq3
A	B	1	1	Seq4

Table 5.2 – Failure Regions Discovered per Case

Probability of a test-case discovering region D, p(D) = 30/50

Probability of a test-case discovering region E, p(E) = 30/50

Probability of a test-case discovering region A, p(A) = 20/50 [dA]

Probability of a test-case discovering region B, p(B) = 20/50 [dB]

F1 = 20 F2 = 20

T = 100 (total points in domain) q = (F1+F2)/T

E (Θ) = ∑ (Probability that the particular case will occur) *
(Probability that the rest of the faults are *un*detected)

E (Θ) = E(Θ) (seq1) + E(Θ) (seq2) + E(Θ) seq3) + E(Θ) seq4)

= {(1- dA)* (1- dB)* (q)} + {(1-dA)* (dB)* (q –F1/T)} +

{(dA)* (1- dB)* (q –F2/T)} + {(dA)* (dB)* (q –F1/T – F2/T)} …….……(1)

= {(30/50 * 30/50) * 40/100} + {(30/50 * 20/50) * 20/100} +

{(20/50 * 30/50) * 20/100} + {(20/50 * 20/50) * 0/100}

= 0.144 + 0.048 + 0.048 = *0.24* ...…….. (2) *E (Θ) Theoretical*

Simulation Result: E (Θ) = 0.2369 ……..….(3) *E (Θ) Empirical*

The close proximity of two values (2) and (3) indicates that the method (1) used to calculate the theoretical E (Θ) value is accurate. To check whether the above described method can be made generally applicable, it was tested against 3 failure regions spread across 3 sub-domains. The results confirmed that the method can be generally applicable.

CHAPTER 6

REDUCTION OF THE METHOD

The method described above can be reduced in form to a formula in the following manner:

$$
\begin{aligned}
E(\Theta) = {} & \{(1- dA)^* (1- dB)^* (q)\} + \{(1-dA)^* (dB)^* (q - F1/T)\} + \\
& \{(dA)^* (1- dB)^* (q - F2/T)\} + \{(dA)^* (dB)^* (q - F1/T - F2/T)\} \\
= {} & q[\{1 - dB - dA + dAdB\} + \{ (dB - dAdB)(1 - F2/T^*q\} + \\
& \{(dA - dAdB)(1-F1/T^*q)\} + \{dAdB\ (1 - F1/T^*q - F2/T^*q)\}] \\
= {} & q\ [\{1 - dB - dA + dAdB\} + \{ dB - dB^*F2/T^*q - dAdB + dAdBF2/T^*q\} + \\
& \{ dA - dAF1/T^*q - dAdB + dAdBF1/T^*q\} + \\
& \{ dAdB - dAdB^*F1/T^*q - dAdBF2/T^*q\}] \\
= {} & q[\ 1 - dBF2/T^*q - dAF1/T^*q] \\
= {} & q - (\ dAF1/T + dBF2/T)
\end{aligned}
$$

This form of reduction when applied to E (Θ) calculation for 3 failure regions in 3 sub-domains yields:

p(A) = 20/50 dA]

p(B) = 15/50 [dB]

p(C) = 10/50 [dC]

	F1	F2	F3
SD1	20/50	0/50	0/50
SD2	0/50	20/50	0/50
SD3	0/50	0/50	20/50

Table 6.1 NOFR, 3 Sub-Domains

F1 = 20, F2 = 20, F3=20

q = (F1+F2+F3)/T

T = 150 (total points in domain)

$$E(\Theta) = \{(1- dA)* (1- dB)* (1- dC)*(q)\} + \{(1-dA)* (1-dB)* (dC)*(q - F3/T)\} +$$

$$\{(1-dA)* (dB)* (1-dC)*(q - F2/T)\} + \{(1-dA)* (dB)* (dC)*(q - F2/T - F3/T)\} +$$

$$\{(dA)* (1-dB)* (1-dC)*(q - F1/T)\} + \{(dA)* (1-dB)* (dC)*(q - F1/T - F3/T)\} +$$

$$\{(dA)* (dB)* (1-dC)*(q - F1/T - F2/T)\} + \{(dA)* (dB)* (dC)*(q - F1/T - F2/T - F3/T)\}$$

$$= q[\ 1 - dCF3/\ T*q - dBF2/T*q - dAF1/T*q]$$

$$= q - \{\ dAF1/T + dBF2/T + dCF3/T\}$$

CHAPTER 7
SUMMARY AND FUTURE WORK

In this paper we have attempted to extend the functionality of E(Θ) beyond the present boundaries and include scenarios so far only calculable through empirical means. We have looked at possible ways of simplifying the lengthy procedure of calculating E(Θ) for software with multiple failure regions and multiple sub-domains and formulated mathematical procedures for calculating the same.

q – (dAF1/T + dBF2/T) represents the formula for calculating E(Θ) for software with two failure regions spread across two sub-domains, while q – { dAF1/T + dBF2/T + dCF3/T} represents the formula for calculating E(Θ) for software with three failure regions spread across three sub-domains. This linear evolution of terms in the formula can be easily generalized for calculating E(Θ) for software with 'n' failure regions spread across 'n' sub-domains.

Derivatives of the thus obtained methods like upper limit calculation of E(Θ) or lower limit estimation of reliability for software after testing were also studied and formulated.

In this paper we have considered cases with only one test case per sub-domain. We plan on generalizing this to "n" test cases per sub-domain and this is one area where we would like to see future work done by including more test cases per sub-domain.

REFERRENCES

[1] Phyllis G. Frankl, Richard G. Hamlet, Bev Littlewood,and Lorenzo Strigini, "Evaluating Testing Methods by Delivered Reliability," IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 24, NO. 8, AUGUST 1998

[2] Marnie.L.Hutcheson, "Software Testing Fundamentals – Methods and Metrics," Wiley Publishing Inc.,2003

[3] Louise Tamres, "Introducing Software Testing," Addison-Wesley, 2002

[4] Rex Black, "Critical Testing Processes," Addison-Wesley, 2004

[5] James.A.Whittaker,"How to Break Software – A Practical Guide to Testing,"AddisonWesley, 2003

[6] Boris Beizer, "Software Testing Techniques," International Thomson Computers Press, 1990

APPENDIX A

ETHETA CALCULATOR LOGIC:

```
//  Implementation of ActionListener interface.
                public void actionPerformed(ActionEvent event)
                {
                        int testCombo[]= new int[4];
                        int fault1,fault2;
                        double remProb1,remProb2;
                        int testPoint[] = new int[2];
                        float freq[]= new float[4];
                        double remainingProb[] = new double[4];
                        int index;
                        double reliability = 0;
                        Random rn = new Random();

                        int tempSub_D1=
(int)((Double.parseDouble(tempSD1.getText())));
                        int tempSub_D2 =
(int)((Double.parseDouble(tempSD2.getText())));
double tempSub_D1FR1 = (int)((Double.parseDouble(tempSD1FR1.getText())));
double tempSub_D1FR2 = (int)((Double.parseDouble(tempSD1FR2.getText())));
double tempSub_D2FR1 = (int)((Double.parseDouble(tempSD2FR1.getText())));
double tempSub_D2FR2 = (int)((Double.parseDouble(tempSD2FR2.getText())));
double q= (tempSub_D1FR1 + tempSub_D1FR2 + tempSub_D2FR1 + tempSub_D2FR2
) / (tempSub_D1+tempSub_D2);
double remProb_Val1 = (tempSub_D1FR1 + tempSub_D2FR1)/(tempSub_D1 +
tempSub_D2);
```

```
double remProb_Val2 = (tempSub_D1FR2 + tempSub_D2FR2)/(tempSub_D1 +
tempSub_D2);

        // Initialize the test combination structure
                        for (int i = 0; i<4 ; i++)
                        {
                                freq[i]= 0;
                                remainingProb[i]=0;
                        }
                        for ( int i = 0; i<5000 ; i++)
                        {
                                fault1 = 0;
                                fault2 = 0;
                                remProb1=remProb2= 1;
                                testPoint[0] = rn.nextInt(tempSub_D1);
                                testPoint[1] = rn.nextInt(tempSub_D2);
                                if ( testPoint[0] <= tempSub_D1FR1)
                                {
                                        fault1 = 1;
                                        remProb1 = 0;
                                }
        if ( testPoint[0] > tempSub_D1FR1 && testPoint[0] <= (tempSub_D1FR1 +
tempSub_D1FR2))
                                {
                                        fault2=1;
                                        remProb2 = 0;
                                }
                                if (testPoint[1]  <= tempSub_D2FR1)
                                {
                                        fault1 = 1;
                                        remProb1 = 0;
```

```
                    }
if (testPoint[1]  > tempSub_D2FR1 && testPoint[1]  <=
(tempSub_D2FR1+tempSub_D2FR2))
                    {
                        fault2 = 1;
                        remProb2 = 0;
                    }
                    index = (fault1*2) + (fault2*1) ;
                    freq[index]++;
        remainingProb[index]= (remProb1*remProb_Val1) + (remProb2 *
remProb_Val2) ;
                }
                for (int i =0 ; i<4 ; i++)
                {
                    reliability  = reliability + ( (freq[i]/5000) *
remainingProb[i] );
                }
                EthetaVal.setText(reliability + " Etheta value");
            }
```

APPENDIX B

USER INTERFACE SNIPPET:

```
class ParameterIP2 implements ActionListener
        {
                JFrame testing2Frame;
                JPanel testing2aPanel;
                JTextField tempSD1; // SD1 = Sub Domain 1
                JTextField tempSD2; // SD2 = Sub Domain 2
                JTextField tempSD1FR1;      // SD1FR1 = Failure Region 1 in SD1
                JTextField tempSD1FR2;      // SD1FR2 = Failure Region 2 in SD1
                JTextField tempSD2FR1;      // SD1FR1 = Failure Region 1 in SD2
                JTextField tempSD2FR2;      // SD1FR2 = Failure Region 2 in SD2

                JLabel EthetaVal;
                JButton CalculateEtheta;

                // Constructor
                public ParameterIP2()
                {
                        // Create the frame and container.
                        testing2Frame = new JFrame("SD & FR Input Frame");
                        testing2Frame.setSize(40, 40);
                        testing2aPanel = new JPanel();
                        testing2aPanel.setLayout(new GridLayout(2,2));

                        //      Add the widgets.
                        addWidgets();
```

```
                    // Add the panel to the frame.
testing2Frame.getContentPane().add(testing2aPanel, BorderLayout.NORTH);

                    // Exit when the window is closed.

        testing2Frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

                    // Show the converter.
                    testing2Frame.pack();
                    testing2Frame.setVisible(true);
                }

        private void addWidgets()
                {
                    // Create widgets.
                    tempSD1 = new JTextField(2);
                    tempSD2 = new JTextField(2);
                    tempSD1FR1 = new JTextField(2);
                    tempSD1FR2 = new JTextField(2);
                    tempSD2FR1 = new JTextField(2);
                    tempSD2FR2 = new JTextField(2);
                    tempSD1.setText("SD1 val");
                    tempSD2.setText(" SD2 val");
                    tempSD1FR1.setText(" SD1FR1 ");
                    tempSD1FR2.setText(" SD1FR2 ");
                    tempSD2FR1.setText(" SD2FR1 ");
                    tempSD2FR2.setText(" SD2FR2 ");

                    CalculateEtheta = new JButton("CalculateEtheta");
                    EthetaVal = new JLabel("EthetaVal", SwingConstants.RIGHT);
```

```
        // Listen to events from Convert button.
        CalculateEtheta.addActionListener(this);

        // Add widgets to container.
        testing2aPanel.add(tempSD1);
        testing2aPanel.add(tempSD2);
        testing2aPanel.add(tempSD1FR1);
        testing2aPanel.add(tempSD1FR2);
        testing2aPanel.add(tempSD2FR1);
        testing2aPanel.add(tempSD2FR2);

        testing2aPanel.add(CalculateEtheta);
        testing2aPanel.add(EthetaVal);


EthetaVal.setBorder(BorderFactory.createMatteBorder(5,5,5,5,Color.cyan));

    }
```

APPENDIX C

TEST RESULTS:

The following tests were performed to validate the simulator performance with five or more sub-domains:

	Flt 1	Flt 2	Flt 3	Flt 4	Flt 5
SD 1	10/20	0/20	0/20	0/20	0/20
SD 2	0/10	5/10	0/10	0/10	0/10
SD 3	0/10	5/10	1/10	1/10	0/10
SD 4	0/20	0/20	2/20	0/20	0/20
SD 5	0/20	0/20	2/20	0/20	1/20

Table 3A.1 – NOFR, 5 Sub-Domains

comb	Fprob(q)	Freq(p)	Prod
0	0.3375	0.7062	0.23834251
1	0.325	0.0096	0.00312
2	0.325	0.0112	0.0036399998
3	0.3125	0.0	0.0
4	0.275	0.0432	0.01188
5	0.2625	2.0E-4	5.2499996E-5
6	0.2625	2.0E-4	5.2499996E-5
7	0.25	0.0	0.0
8	0.2125	0.1	0.02125
9	0.2	0.0014	2.8E-4
10	0.2	6.0E-4	1.20000004E-4
11	0.1875	0.0	0.0
12	0.15	0.0056	8.4E-4

13	0.1375	0.0	0.0
14	0.1375	0.0	0.0
15	0.125	0.0	0.0
16	0.2125	0.0988	0.020995002
17	0.2	0.0010	2.0000001E-4
18	0.2	8.0E-4	1.6E-4
19	0.1875	0.0 0.0	
20	0.15	0.0054	8.1000006E-4
21	0.1375	0.0 0.0	
22	0.1375	2.0E-4	2.75E-5
23	0.125	0.0	0.0
24	0.0875	0.0148	0.001295
25	0.075	2.0E-4	1.50000005E-5
26	0.075	2.0E-4	1.50000005E-5
27	0.0625	0.0	0.0
28	0.025	4.0E-4	1.0E-5
29	0.0125	0.0	0.0
30	0.0125	0.0	0.0
31	0.0	0.0	0.0

E(theta) =0.30310506

	Flt 1	Flt 2	Flt 3	Flt 4	Flt 5
SD 1	10/20	0/20	0/20	0/20	0/20
SD 2	0/10	5/10	0/10	0/10	0/10
SD 3	0/10	0/10	4/10	0/10	0/10
SD 4	0/20	0/20	0/20	2/20	0/20
SD 5	0/20	0/20	0/20	0/20	5/20

Table 3A.2 – NOFR, 5 Sub-Domains

comb	Fprob(q)	Freq(p)	Prod
0	0.3375	0.7152	0.24138
1	0.325	0.0478	0.015535
2	0.325	0.0176	0.00572
3	0.3125	0.0016	4.9999997E-4
4	0.275	0.0396	0.0108900005
5	0.2625	0.0022	5.775E-4
6	0.2625	0.0014	3.6749998E-4
7	0.25	2.0E-4	5.0E-5
8	0.2125	0.044	0.00935
9	0.2	0.0032	6.4E-4
10	0.2	8.0E-4	1.6E-4
11	0.1875	0.0	0.0
12	0.15	0.0020	3.0E-4
13	0.1375	4.0E-4	5.5E-5
14	0.1375	0.0	0.0
15	0.125	0.0	0.0
16	0.2125	0.1022	0.021717502
17	0.2	0.0074	0.00148
18	0.2	0.0026	5.2E-4
19	0.1875	2.0E-4	3.7499998E-5
20	0.15	0.0044	6.6E-4
21	0.1375	2.0E-4	2.75E-5
22	0.1375	0.0	0.0
23	0.125	0.0	0.0
24	0.0875	0.0052	4.55E-4
25	0.075	2.0E-4	1.50000005E-5
26	0.075	2.0E-4	1.50000005E-5
27	0.0625	2.0E-4	1.25E-5
28	0.025	0.0012	3.0000001E-5
29	0.0125	0.0	0.0
30	0.0125	0.0	0.0
31	0.0	0.0	0.0

E(theta) =0.31049496

	Flt 1	Flt 2	Flt 3	Flt 4	Flt 5
SD 1	10/20	0/20	5/20	0/20	0/20
SD 2	0/10	5/10	0/10	1/10	2/10
SD 3	0/10	2/10	2/10	1/10	0/10
SD 4	2/20	4/20	2/20	2/20	5/20
SD 5	0/20	0/20	1/20	5/20	1/20

Table 3A.3 – NOFR, 5 Sub-Domains

comb	Fprob(q)	Freq(p)	Prod
0	0.575	0.5248	0.30176
1	0.5625	0.0652	0.036675
2	0.4875	0.0552	0.02691
3	0.3875	0.0060	0.002325
4	0.45	0.075	0.03375
5	0.35	0.0092	0.00322
6	0.3625	0.0092	0.0033350002
7	0.2625	4.0E-4	1.0499999E-4
8	0.4625	0.0868	0.040145002
9	0.3625	0.0068	0.0024650001
10	0.375	0.0076	0.00285
11	0.275	0.0	0.0
12	0.3375	0.0108	0.003645
13	0.2375	0.0	0.0
14	0.25	8.0E-4	2.0E-4
15	0.15	0.0	0.0
16	0.425	0.0968	0.04114
17	0.325	0.0118	0.003835
18	0.3375	0.0096	0.0032400002
19	0.2375	4.0E-4	9.4999996E-5
20	0.3	0.0062	0.0018600001

21	0.2	4.0E-4	8.0E-5
22	0.2125	0.0010	2.1250002E-4
23	0.1125	6.0E-4	6.7500005E-5
24	0.3125	0.0132	0.004125
25	0.2125	6.0E-4	1.2750001E-4
26	0.225	6.0E-4	1.3500001E-4
27	0.125	0.0	0.0
28	0.1875	8.0E-4	1.4999999E-4
29	0.0875	2.0E-4	1.7499999E-5
30	0.1	0.0	0.0
31	0.0	0.0	0.0

E(theta) =0.51247