ISSN: 2321-7782 (Online)
e-ISJN: A4372-3114
Impact Factor: 6.047

Volume 5, Issue 6, June 2017

International Journal of Advance Research in Computer Science and Management Studies

Research Article / Survey Paper / Case Study

Available online at: www.ijarcsms.com

A Code Level Criticality Analysis Model for Test Path Generation

Kavita[1]
Research Scholar,
Computer Science and Application Department
MDU, Rohtak – India

Dr. Priti[2]
Assistant Professor
Computer Science and Application Department
MDU, Rohtak – Nigeria

Abstract: ***While measuring the software system for quality or the reliability factor, one of the critical aspects is software fault. In this paper, Software criticality analysis based segmented backtracked method is proposed. The complete process model defined in this work is divided into two major stages. On the first stage, the software system analysis is done to identify different inputs for test cases. In this stage, the division of a software system is done in smaller segments and identifies the expected criticalities in each segment. Along with these criticalities, the frequency and the criticality type are also identified. Finally, all these factors are combined to generate the weights for each segment. In the second stage, the backtracked analysis method is applied to generate path sequence. This stage specific method identifies the possible path sequence and estimate the individual stage cost. While moving to the next stage, the aggregate measure is applied to identify the combined cost. The final cost is considered as the path cost inclusive to individual criticality handling. The working model is applied to different code segments. The analytical result shows that the model has identified the test sequence effectively.***

Keywords: Backtracked; Criticality Analysis; Segmented; Structured.

I. INTRODUCTION

To ensure the reliability of a software system, it is required to apply different testing methods over it. These testing techniques can be applied at run level or at structure level. As some testing method is applied on a code segment or software system, each of the statement or section is analyzed individually. But each of the test cases does not have equal importance. Path testing [1] [2] [4] [5] [6] [7] is about to identify the process sequence of test case execution. While performing the test sequence selection there is the requirement to analyze the software system under different parameters. One such parameter is the coverage of software systems. The coverage here means that each of the software segments must be analyzed. The repetition and overlapping should also be avoided while dividing the segments. Another aspect is to set the priorities parameter or constraint. Some of these parameters include fault based analysis, module interaction measure, dependency, path flow observation, etc. Based on this parameter, the test cases are identified for each segment and relatively test data is generated. The generation of test data can be done manually or in automated way based on the test case [8] [10] [11] [12] [14]. In this work, software criticality is considered as the prior measure based on which software system analysis is done and priority to each test case is obtained. In this section, the basic constraints and phenomena associated with path testing are discussed.

A. Test Data Generation

After generating the test cases for the particular code segment or software module, the next work is apply the test data [5] [7] [10] [17] [18] on these test cases. This data is defined by the particular methodology specification. Such as Boundary Value analysis method able to generate the minimum and maximum limit of feasible data to the system. Test data must be defined by flow specific analysis. It means the test data of one module can be the output of another module. Such kind of observations can

be taken either by using dynamic, automated methods or by generating some expected evaluation measure. The behavior specific and intellectual observation are applied to identify the problem in the functionality or behavior of software system. The structure and constraint specific analysis is required to identify the most appropriate test data. The reliability of test system depends on its validation from each possible data. Effective Test-data generation ensures this validity. The relational observation with structural analysis also increases the criticality of this test system.

B. Boundary Value Analysis

It is the limit analysis method applied while generating the test data for a particular test case. Boundary value is about to identify the upper and lower peaks of accepted feasible values for a software system or software segment. In black box testing, the run time observations are considered in this limit constraint. In some structural test cases, the limits also applied to track the feasible accepted data at an earlier stage. If these cases and data are identified at an earlier stage, the preventive code segments can be applied to intimate the user by generating some error message [7] [10] [17].

C. Structured Analysis

The base of path testing is to generate the complete structural flow of software systems. If the software system is already defined with relative modules, the structure is clearly defined. In complex integrated system, it is required to divide the software system in smaller segments. The connectivity and interaction between these segments are also defined to generate the structured flow. The structural observation is essential for both black box and white box testing. In most of software system, the structured flow of black box testing is also applied in white box testing methods [1] [2] [5] [6] [7] [8]. The clarity of this structure generation affects the efficiency of test case observation. Individual segment and the interactive segment analysis are applied to analyze the complete software system.

D. Test Case Prioritization

After generating the test cases, the next work is identifying the criticality of each test case separately under the light associated parameter. This priority or weight assignment is required to identify the essential and critical test cases. The dependency between the test cases is also identified to generate the effective test sequence. This prioritization is generally based on the static measures or weight function applied to each test case. These weights can be identified using one or more parameters. In this work, a faith based prioritization method is applied [11] [15] [17] [18] [20].

In this paper, a weight driven segmented backtrack method is defined to identify the optimized test sequence. The work model has defined a weight function to assign the cost of each test case under criticality vector. Later on, the test process sequence is identified to improve the software reliability. In this section, different constraints associated with path testing are discussed. In section II, the work provided by earlier researchers is also discussed. In section III, the proposed weight function is presented and discussed. In section IV, the backtrack algorithm is defined by stage specification. In section V, the result observation applied in sample program is evaluated. In section VI, the conclusion obtained from the work is presented.

II. RELATED WORK

Different research included their contribution for optimizing the path testing and generating the more effective path. Some of the work defined by earlier researchers is described in this section. Hina Sattar [1] has identified the challenges in path testing and presented a scheme for automated testing. Author identified the process overhead in testing and categorizes the structural and the procedural challenges in the testing methods. Author also focused the work on environmental constraints and describes the automated phenomenon to generate the test sequence with step specific derivation. Author [2] applied the black box testing path observation in white box testing. The black box testing is applied to web site to generate the structural observations which are later on applied on the same projects while performing the test procedure. The usability provided the early traversing of the path so that the cost and efforts are reduced. The navigational structure based estimates provided the effective coverage of

critical modules. Author [3] designed a tool to generate the path test sequence with criteria specified. The complexities of branch structure estimation and the determination of the relative inadequate problems are also provided by author. The tool defined a dynamic observation of program to generate the test cases and generated a test sequence with parameter specific observation. Qingfeng [13] used linear dependency observation in the control flow graph for test path generation. The method covered all feasible paths for maximum coverage. Martin [14] defined an experimental study on different path estimation methods for different programs. Author defined the test metrics for structure specific estimation and to achieve maximum coverage. Khalil [19] defined a criteria specific path testing with round trip observation. The traversal specific observation with different assumption is defined for tree construction algorithm.

After identifying the dynamic observation of programs and the challenges in the test path generation, the next work was to optimize the test path generation results. Biswas [4] has optimized the process using ACO (Ant Colony Optimization) method. This swarm based approach prioritizes the test cases based on test data observation and relative to the domain specific sequence the coverage adaptive sequence path is generated. The optimization method has resolved the redundancy situation and provided a more effective test path generation. Jin-Cherng [6] used the Integer programming as an optimization method to measure the effective test path. Author applied the coverage criteria based test path generation to cover different test branches and with test data specification. Author [9] used the constraint reasoning method for path optimization. A flow specific modeling with rejection approximation is provided by the author. The propagation and refutation are analyzed by the researcher at the same time. Yang [12] used the genetic modeling for test path generation. The structural approximation with parameter specification is defined. The stage based architecture is defined to estimate the reliability and effective path generation with architectural modeling. Imran [15] explored the challenges and key parameters of genetic based optimization of path testing. Author applied different experimentation on different data sets and parameters with different population size to get real time observation. Jin-Cherng [16] used the target specific genetic optimization of test case generation and test sequence optimization. Author iteratively defined the process of case evolving optimization.

Another aspect of path testing is the test data generation based on which quality of test procedure depends. Automated generation of trial data and its selection from pool is also a research query. Takeshi [5] provided at work on data selection and quality estimation with specification of test cases and test branches. A linear and essential method is defined to generate test data with redundancy elimination. Test data generation can be also requires optimization to improve this procedure. One such improvement is provided by Juhi [7] using Genetic approach. Author used it for aspect oriented programming with observation of control flow graph. A framework driven method is defined for test data generation and optimization. Gentiana[10] also used the evolutionary method for test data generation. Author applied the level and code structure specific approximation for method driven estimation. Author applied distance metrics for optimization measures. Xibo [17] also used a genetic process modeling for test data generation. The problem constraint based modeling and operator specific observations are set for test data generation. A real process formation method with key algorithm is specified by the source. Mohapatra [18] generated the data design and applied data sampling using genetic modeling. Author tried to reduce the effort and cost at an earlier stage of applying the meaningful data. A category partition based genetic is applied for test data generation.

Reliability is also the major constraint of Path testing covered by Chao-Jung [8]. An inter-factor analysis based reliability method is defined for extended analysis of modular resources. An incorporating path is generated with framework estimation in terms of closed circuit, branch, structure, etc. Along with path generation, the author provided the effective resource allocation. Chao-Jung [11] integrated the control flow analysis for reliability estimation using three methods. These methods include sequencing structure analysis, branch specific and loop driven observation. The validations are also used to distinguish the work effectiveness.

III. CRITICALLY BASED WEIGHT FUNCTION

The contribution of this report is to develop a weighted method for assigning the priority to code segments. Code level criticality is considered here as the basic phenomenon for priority assignment. Code criticality is the integrated measure analyzed based on the type of code criticality, type of test case criticality and the frequency of associated problem in code segment. Each of these constraints are applied individually and later on formed under a formulated formula to generate the aggregative weights. At the earlier level, after generating the code segments, the first study is to break down the code or the module functionality and put it some score. The grading of a module is founded on different parameters including

- Frequency of Module Usage
- Purpose of Module
- Dependency of Module
- Impact of Module

All these criteria are observed collectively to assign the scoring to the modules. The scoring is here applied between 1 and 5. 1 is assigned for a low priority module which is used rarely whereas 5 is the high priority module with high criticality.

Table 1 : Module Functionality Scoring

Module Code	Functionality	MScore
Seg 1	Control Placement	1
Seg 2	Interface Easiness	1
Seg 3	Search Work	3
Seg 4	Save Button Process	4
Seg 5	Delete Button Process	5
Seg 6	Backup	4
Seg 7	Restore	5

After placing the module scoring, the next level is to get the test cases appropriate to the individual module. This stage has identified the number of test cases and criticality of each test instance

Table 2 : Module Criticality Estimation

Module Code	Functionality	MScore	Freq (F)	Criticality (C)	FXC
Seg 1	Control Placement	1	1	2	2
Seg 2	Interface Easiness	1	2	1	2
Seg 3	Search Work	3	1	1	1
Seg 4	Save Button Process	4	3	3	9
Seg 5	Delete Button Process	5	4	4	16
Seg 6	Backup	4	2	3	6
Seg 7	Restore	5	4	4	16

This extracted scoring under frequency and criticality is considered as the aggregate measure which is later on applied with static priority range specification measure. In this the scoring is measured with the relative priority aspect specification. The priority aspect defined in the work is indicated in table 3.

Table 3 : Priority Aspect Derivation

NXS	Priority
1-5	1
5-10	2
10-15	3
15-20	4
20-25	5

Grounded on this observation, the final priority to the modules in the program flow or the software system is given. The actual priority derivation applied in the work is indicated in table 4.

Table 4 : Priority Assignment

Module Code	Functionality	MScore	Freq (F)	Criticality (C)	FXC	Priority
Seg 1	Control Placement	1	1	2	2	1
Seg 2	Interface Easiness	1	2	1	2	1
Seg 3	Search Work	3	1	1	1	1
Seg 4	Save Button Process	4	3	3	9	2
Seg 5	Delete Button Process	5	4	4	16	4
Seg 6	Backup	4	2	3	6	2
Seg 7	Restore	5	4	4	16	4

Founded on this priority assignment, the risk exposure of the impact on the software quality is valued. The risk rating is also founded on the criticality vector observation and it is indicated in table 5.

Table 5 : Risk Evaluation

Module Code	Functionality	MScore	Freq (F)	Criticality (C)	FXC	Priority	RE= Score * Priority
Seg 1	Button Placement	1	1	2	2	1	1
Seg 2	GUI design	1	2	1	2	1	1
Seg 3	Working of Search Button	3	1	1	1	1	3
Seg 4	Working of Save Button	4	3	3	9	2	8
Seg 5	Working of Delete	5	4	4	16	4	20
Seg 6	Backup	4	2	3	6	2	8
Seg 7	Restore	5	4	4	16	4	20

This estimated risk exposure is thought as the final standard based on which path test sequence is identified. The associated process algorithm and the experimentation are shown in the next sections.

IV. Test Path Sequence Generation

The major contribution of this report is to analyze the software system criticalities to assign the weight and generate the cost efficient path. The cost or the risk exposure assignment for each module is already defined in section III of this paper. The final process defined in this study is to apply the algorithmic process for optimized sequence generation. The proposed work model can be employed to a complex software system as well as form a smaller design. The method takes the clarity in terms of module specification based on which the task can be divided into smaller units. Once the partitioning is built out, the control flow graph will be obtained. This modular dependency based graph will be considered as the primary input to the algorithmic process. A composition of the process phase is defined as the individual code segment which is being examined for weight assignment and for sequence generation. The basic model defined in the presented work is depicted in image 1. After setting the weights to modules, the next work is to identify the linked test cases based on the problem identification. The code level criticalities are identified in this stage. The identification of different sort of defects or the bugs in each part is placed in this stage. At this level, the fault driven observations are done in terms of error frequency, fault criticality and the usability of code.\

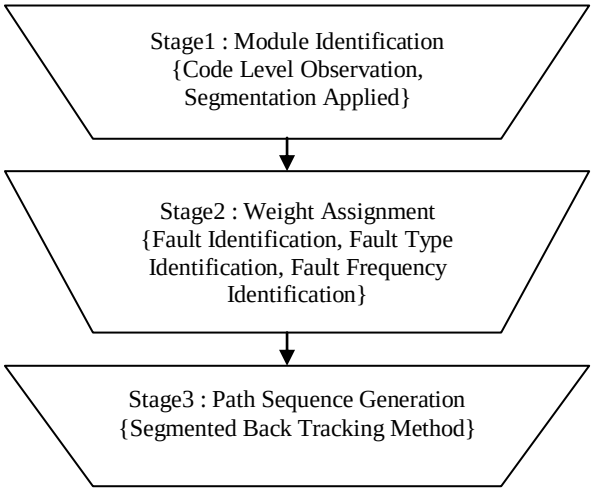

Figure 1: Proposed Model

After weight assignment, the final work is to generate the course sequence based on risk factor evaluation at each level. This method has been used the segmented software system in opposite order and identified the criticality at each layer individually and aggregated. This aggregate composition formed the idea of software system cost so that the actual optimized path will be obtained. The path generation algorithm proposed in this study is demonstrated in table 6.

Table 6 : Path Sequence Algorithm

```
1.  FaultPathAlgorithm(Modules,TestCases)
    /*A software project is the collection of software modules defined with relative test cases */

    {

2.  For i=1 to Modules.Count
    [Analyze each of the Module under Fault Measure to generate optimize path]

    {

3.  For j=1 to Modules (i).TestCases.Length
    [Process All Test Cases associated to each module]

    {

4.  If(Type(Module(i),TestCases(j))=”Warning”))
    [Check for Module type relative to the associated error and test case and relatively assign priority]

    {

5.  Set TestCases(j).Priority=Low
    }

6.  Else If(Type(Module(i),TestCases(j))=”Error”))
    [Check for Module type relative to the associated error and test case and relatively assign priority]

    {

7.  Set TestCases(j).Priority=Medium
    }

8.  Else If(Type(Module(i),TestCases(j))=”Failure”))
    [Check for Module type relative to the associated error and test case and relatively assign priority]

    {

9.  Set TestCases(j).Priority=High
    }
```



```
10. Cost(j)=Size(Module(i))* TestCases(j).Priority * BaseCost
    [Estimate the Cost of Each Module and relative test cases]
    }

11. For i=TestCases.Count to 1
    [Apply Backtracked Dynamic Programming for Cost Estimation]

    {

12. LevelTestCases=LevelCases(TestCases)
    [Identify the test cases present at particular level]

13. For j=1 to LevelTestCases.Count
    [Process All the Test cases present at particular level]

    {

14. for k=level  to Module.Count
    [Perform the aggregative cost analysis]

    {

15. levelcost=GetMinimum(Module(k))
    [Identify the minimum cost at particular level]

16. TCost(k)=TCost(k)+levelcost+ LevelTestCases(j).cost
    [Obtain the aggregative cost]

    }

17. Optimizedcost=Minimum(TCost)
    [Identify the aggregative minimum cost]

    }

18. Return Optimizedcost
}
```

The algorithm specified in table 6 signifies the process stages of this study. The algorithm here shows that the input is here taken in terms of software modules and associated test cases for each stage. At the initial level, the modules and test cases are analyzed in terms of associated error. The module criticality is here analyzed in terms of warning, error and the failure. Based on this, the priority of each test case is assigned. After identifying the test case priority, the cost estimation is also done.

In second phase of this algorithmic model, the dynamic programming approach is applied. This method is applied in backtracked mode. The reverse module tracking is here applied with cost evaluation. The test cases are identified at particular module level and aggregated to the cost to the last module. This process is repeated till the first module. The algorithm has identified the optimized cost and the path.

V. RESULTS

In this segment, the code segment based work model implementation is limited. The sample programs are needed for the work implementation. These vectors are defined here under different vectors. The Segmented Aggregative parameters along with Code Path Testing parameters are defined hereunder

Table 5 : Sample Code

```
Void main()
{
Int a,b,c;

Printf("Enter number");
Scanf("%d%d",&a,&b);
C=sqrt(a)/b;
Printf("%d",c);
}
Values
```

As the Segmented Aggregative procedure is used, the results obtained from the work are presented in table 6.

Table 6 : Test Cases

1.	Variable A Assigned Some Value;
2.	Variable B Assigned Some Value;
3.	Variable C Assigned Some Value;
4.	Check A for 0
5.	Check B for 0
6.	Check A for –ve `
7.	Check for input specifier for A,B
8.	Check for Expression Result of C
9.	check for output Specifier for C

One of the usual approaches for the priority designation is the priority based on the distance from the goal state. Agreeing to this method, the test case, nearer to the goal is assigned with highest priority as it is expected that the special event is asked to retest. In the same direction as the distance from the goal stage will be increased, the overall test case will be lessened. The parameters of the Segmented Aggregative process are presented hereunder

Table 7 : Test Priortization

Here in given program Test Case 5 will return failure as of B is 0
Here Test Case 6 will return Error if A is I –ve
Here Test Case 9 will return error if any of Case 5 or 6 is false
Case 10 will not return correct result if specifier is wrong

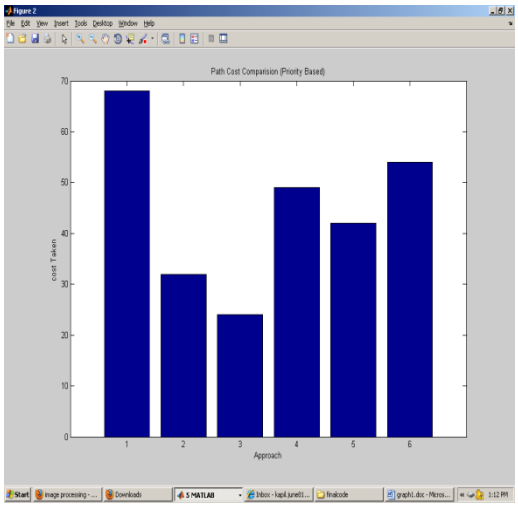

Figure 2: Analysis under Prioritization Assignment Approaches

Here figure 2 is demonstrating the output of presenting work under different prioritization assignment approaches. The approaches taken here are random priority assignment, ascending order or descending order. The figure is shown the random

prioritization to the test case is effective as it presents more chances of test case selection so that the optimum test sequence can be obtained.

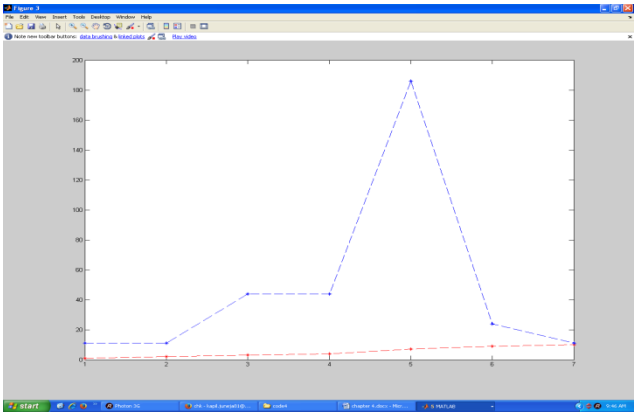

Figure 3: Test Path Generation

Here figure 3 is showing the generated test path of the oeuvre. The test path selection obtained along with associated test cost is indicated here.

VI. CONCLUSION

In this newspaper, an advance to the test path generation is determined by using the improved Segmented Aggregative approach. The oeuvre is limited under the test sequence prioritization approach. The outcomes indicate that the random prioritization approach is more effective than ordered prioritization to identify the optimal path sequence. The analysis is here doe in terms of optimum cost estimation and the performance appraisal in terms of time. The random prioritization approach is effective under both these elements.

References

1. Hina Sattar," Automated DD-Path Testing: A Challenging Task in Software Testing", 978-1-4799-5421-6/14 © 2014 IEEE.
2. Rajiv Chopra," Reusing Black Box Test Paths For White Box Testingof Websites", 978-1-4673-4529-3/12@ 2012 IEEE.
3. T. K. Wijayasiriwardhane," An Automated Tool to Generate Test Cases for Performing Basis Path Testing", The International Conference on Advances in ICT for Emerging Regions - ICTer2011 : 095-101 , 978-1-4577-1114-5/11 ©2011 IEEE.
4. Sumon Biswas," Applying Ant Colony Optimization in Software Testing to Generate Prioritized Optimal Path and Test Data", 2nd Int'l Conf. on Electrical Engineering and Information & Communication Technology (ICEEICT) 2015, 978-1-4673-6676-2115© 2015 IEEE.
5. TAKESHI CHUSHO," Test Data Selection and Quality Estimation Based on the Concept of Essential Branches for Path Testing", IEEE TRANSACTIONS ON SOFTWARE ENGINEERING 0098-5589/87/0500-0509© 1987 IEEE.
6. Jin-Cherng Lin," Zero-One Integer Programming Model in Path Selection Problem of Structural Testing", 0730-3157/89/0000/0618@1989 IEEE.
7. Juhi Khandelwal," Approach for Automated Test Data Generation for Path Testing in Aspect-Oriented Programs using Genetic Algorithm", International Conference on Computing, Communication and Automation (ICCCA2015) ISBN:978-1-4799-8890- 7/15 ©2015 IEEE.
8. Chao-Jung Hsu," An Adaptive Reliability Analysis Using Path Testing for Complex Component-Based Software Systems", IEEE TRANSACTIONS ON RELIABILITY 2011, 0018-9529 © 2011 IEEE.
9. Arnaud Gotlieb, "Constraint reasoning in Path-oriented Random Testing", Annual IEEE International Computer Software and Applications Conference, 0730-3157/08 © 2008 IEEE.
10. Gentiana Ioana Latiu," Automatic Test Data Generation for Software Path Testing using Evolutionary Algorithms", 2012 Third International Conference on Emerging Intelligent Data and Web Technologies 978-0-7695-4734-3/12© 2012 IEEE.
11. Chao-Jung Hsu," Integrating Path Testing with Software Reliability Estimation Using Control Flow Graph", 978-1-4244-2330-9/08 ©2008 IEEE.
12. Yang CAO," An Approach to Generate Software Test Data for a Specific Path Automatically with Genetic Algorithm", 978-1-4244-4905-7/09©2009 IEEE.
13. Du Qingfeng," An Improved Algorithm for Basis Path Testing", 978-1-61284-109-0/11 ©2011 IEEE.
14. MARTIN R. WOODWARD," Experience with Path Analysis and Testing of Programs", 0098-5589/80/0500-0278@1980 IEEE
15. Irman Hermadi,"Genetic Algorithm Based Path Testing:Challenges and Key Parameters", 2010 Second WRI World Congress on Software Engineering 978-0-7695-4303-1/10 © 2010 IEEE.
16. Jin-Cherng Lin," Using Genetic Algorithms for Test Case Generation in Path Testing", 1081-7735/00@2000 IEEE.

17. Wang Xibo," Automatic Test Data Generation for Path Testing Using Genetic Algorithms", 2011 Third International Conference on Measuring Technology and Mechatronics Automation 978-0-7695-4296-6/11© 2011 IEEE.
18. Debasis Mohapatra," Automated Test Case Generation and Its Optimization for Path Testing Using Genetic Algorithm and Sampling", 2009 WASE International Conference on Information Engineering 978-0-7695-3679-8/09© 2009 IEEE.
19. May Khalil," On the Round Trip Path Testing Strategy", 2010 IEEE 21st International Symposium on Software Reliability Engineering 1071-9458/10 © 2010 IEEE.

