Effective Path Selection to Estimate Software Reliability

B.M.Gouthami
PG student,
Rajalakshmi Engg College,
Chennai, Tamilnadu,India.

P.Kumar
Associate Professor,
Rajalakshmi Engg College,
Chennai. Tamilnadu, India.

ABSTRACT

With the increase in use of software system for complex applications there is a growing need for software engineers to prove its reliability and assure its quality. Software reliability and quality assurance has high correlation with failure intensity. Failure can be best analyzed by white box testing. Basis path testing is an important white box testing approach, as the coverage in path testing is high it is directly proportional to its reliability. We propose a frame work to test the path for a structural language. Three major elements of structural language include sequence, branches and loop structures. Evaluating the reliability of each node in turn helps in evaluating the path reliability. Further, software reliability is achieved by correlating the reliability of each unique path followed by the system. Specifically, higher the path coverage higher is the accuracy of reliability. Hence the proposed system helps in evaluating the software reliability based on path testing for a system developed with structural language. Also critical nodes could be identified with which the critical paths are estimated hence to prioritize fault correction.

Keywords

Software reliability, Path testing

1. INTRODUCTION

Software Reliability is used to determine the fault free operation of a system. Most common method used to determine the reliability is using classical reliability model. To predict the reliability of a system using reliability models large set of test data is essential. Alternate approach to evaluate reliability of a system is using path reliability. It is evident that path reliability plays an important and critical role for determining the reliability of a system.

Software reliability is an open problem. The task of software reliability is to achieve a fault free system. The challenge is to accurately evaluate the reliability of the system and use the result to effectively deliver a better system with least errors. The problem of reliability prediction may vary from the actual reliability of the system. The goal of this system is to effectively select the possible paths of the system using graph data structure. The selected paths are tested to evaluate the path reliability with which additive approach is used to evaluate the system reliability. Optimized solution for the test results is used to improve the reliability of the system.

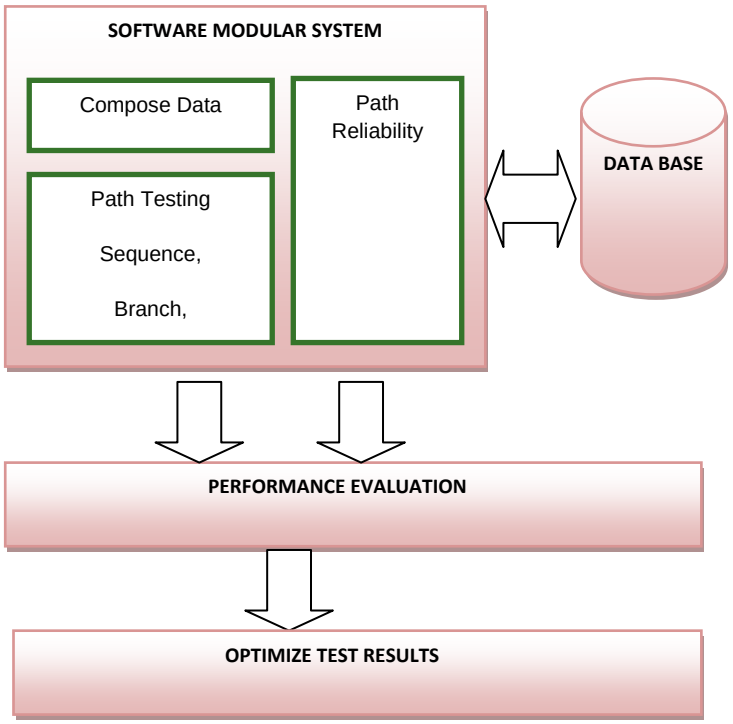

Fig .1Architecture diagram

Fig.1. Shows that modular system consists of sequence, branch and loops in it. This system is converted into flow graph which consists of graphs and edges. From the obtained flow graph of the system all the paths are identified with which path testing is performed. This base work is composed as one component and the resultant data is stored in a data base. From the result reliability parameters are evaluated. Finally the obtained results are optimized.

Software reliability

Software reliability is defined as the failure free operation of the software system. Reliability of the system could be measured based on the errors produced by it. The misbehavior of the system could be identified if it is tested. Hence in order to evaluate the software reliability failure data of the system is required. Failure data could be best obtained if the system is tested completely and consistently.

Path testing

Path testing is the type of white box testing. In white box testing in the internal structure of the system is considered for

testing. Since there is a high correlation between the software reliability and white box testing, path testing is used to evaluate the reliability of the system. In order to obtain the accurate reliability all the paths in the system should be tested. In order to do so all the possible paths in the system is to be identifies. Once all the paths are identified each path is tested to evaluate the path reliability with which the system reliability is evaluated.

2. REVIEW OF RELATED WORKS

S. S. Gokhale [1], have proposed a framework to evaluate the architecture of the system based on the reliability of the component and its application architecture. Architecture is a framework that specifies component and interaction between components. Analysis based on architecture is classified into path based and state based. In path based approach the reliability is estimated through several execution paths. Here the transition between nodes is the probability of transition between node $_{i-1}$ to node $_{i}$. In state based approach control flow graph is mapped to a state space model. Here transition between nodes is the transition from one state to another. They have faced two issues, optimization of test results and second issue, considering interface failure to evaluate software reliability. The assumption here is that only one component should be executed at a time.

K. Go.seva-Popstojanova and K. S. Trivedi [2] have proposed that there is a need to model an approach that is capable of considering the architecture and evaluating the reliability by taking into account the interaction between components, reliability of components and reliability of interfaces with other components. The requirements of architecture based approach is to identify the module i.e. smallest executable unit of system, identify or model the architecture of the system, identify failure behavior and combine failure with the architecture of the system. There are three ways to combine architecture of software with its failure behavior as state based approach, path based approach and additive approach. In state based approach the control flow graph is assumed to represent the states of the system. In path based approach system reliability is computed by considering all the possible execution paths. In additive approach architecture of the system is not explicitly considered. There are three major issues arrived, there is a trade off defining the size of the component, interface reliability is not considered and transition probability issues are not considered.

M.Xie, G.Y.Hong and C.Wohlin, [3] have proposed practical method to predict the software reliability using software reliability model. In order to predict the reliability, large amount of test data is essential. In most cases development team will be interested to estimate the reliability in the early phases of development. Since most of the large software systems are developed in such a way it is the modification of existing system, the failure data of the previous system could be used to predict the parameters of the new system. Two major parameters to evaluate reliability are total number of initial faults and fault detection rate in testing. Hence the software reliability of the new system could be predicted from the test data of similar systems.

Y. K. Malaiya, M. N. Li, J. M. Bieman, and R. Karcich, [4] have proposed a way that quantifies the degree of thoroughness of testing. Relationship between testing time, coverage and reliability is modelled. Test coverage is measured in terms of statement coverage, branch coverage, c-use coverage and p-use coverage. The c-use is a set that contains point where the variable is defined or modified followed by the point where it is used for evaluation .The p-pair is a set that contains the point where the variable is defined or modified followed by the point which is the destination of branching statement where it is used as the predicate. It is proved that if all paths in the program have been exercised, then all the p-use must have been covered similarly all p-use covers ensures branch cover etc. Test data is used to prove that defect coverage implies test coverage and in turn helps the reliability of the system. The only issue is that it is impossible to predict the remaining defects with respect to the coverage.

C. Y. Huang and C. T. Lin, [5] proposed a new approach in software reliability consideration. The general assumption is that the detected faults are immediately corrected in the case of a system. But in reality it is not true. Detecting fault is one issue and correcting them is another. In this paper the faults are classified into two types, dependent faults and independent faults. Mutual independent faults are those that can be removed directly. Dependent faults could be removed only if leading faults are removed. Higher proportion of dependent faults affects the reliability of the system. Next issue is the optimum release time of the product. It depends on two issues; one is reliability requirement of the product and second issue is the budget. As the reliability requirement increases, time taken to achieve the reliability also increases hence increases the cost involved to achieve reliability. If the faults are dependent faults then more time is consumed to achieve reliability hence more cost.

Roberto, S.Russo and K. S.Trivedi [6] proposed an approach to quantitatively identify the most critical component in order to best assign the resource to them. A relation between software reliability and testing time allocation is modelled. It is stated that reliability increases as the software is improved. The concept of visit counts is used to represent the criticality of the component based on the combination of depth of the component, number of user functions called and number of test cases written for that component. System reliability is calculated by considering the individual reliability of the component along with the number of visits to each component. Proposed future work of the author is to extend the system to support concurrent systems.

Lance fiondella and Swapna S.Gokhale [7] presents an optimization framework that considers the contribution of each component to the system reliability.Hence to determine the amount of effort to be allotted to each component to attain the required reliability with minimal effort. Here the

reliability is governed by two factors, system reliability and relation between testing effort applied to the component and its reliability. One factor that influences the reliability of the component is the effort spent on testing that component. Effort spent on each component is based on the complexity of the component and its implementation technology. The reliability of the component with high criticality is set to maximum. The over all effort and time available to resolve the entire system is then distributed across the components based on there importance in the architecture. By reducing the target reliability of the particular component and its fault detection rate the overall effort spent on that component could be achieved optimally.

Chao-Jung Hsu and Chin-Yu Huang [8] proposed an adaptive framework to incorporate path testing technique into reliability estimation for modular software system. Path testing is a white box technique that considers sequence, branch and loop to evaluate the test results. Path testing is a type of white box testing that is modeled to estimate the reliability based on the software structure. The assumptions of the proposed framework is that, the software system is designed using structural or modular philosophy, all modules are physically independent of each other, when faults are found and removed no new faults is introduced, the transfer of control between modules can be described by a markov process.

3. PROBLEM FORMULATION

Software reliability specifies the fault free operation of the system. In order to evaluate the software reliability white box testing approach is used specifically path testing. To perform path testing all the possible paths present in the system are to be identified. To achieve this, initially the given system is to be converted to its equivalent control flow graph. A control flow graph is the one that is composed on nodes and edges. Control flow graph is modified to compose or decompose nodes were ever required. Using the control flow graph all the possible paths present in the system are found using the proposed path identification algorithm. All the identified paths are to be tested to find faults. Path testing results are used evaluate the path reliability with which the system reliability can be evaluated. Next goal is to optimize the test results. To optimize the test results critical nodes and critical paths are to be identified. Critical nodes are nodes that are repeated in maximum number of paths. Critical paths are paths that contain maximum number of critical nodes. Criticality of the paths could be used to fix the on time bugs.

4. EFFECTIVE PATH SELECTION AND OPTIMIZATION

The steps involved in effective path selection and optimization is as follows

Compose data

This module is concerned with composing the source code of the given system. Source code of various sizes can be included. Loaded file is linked with the appropriate library files. Then loaded file is interpreted for errors present in it. With the overall errors the reliability of the system cannot be predicted because the errors does not relate to its complexity. Size of the file, time of execution (calculated in ms) and number of errors are noted.

A sample of code is given below

```
#include <stdio.h>
 main()
{
 int number, sum = 0, temp, remainder;  ---------- node 1
 printf("Enter a number\n");      ---------- node 2
 scanf("%d",&number);       ---------- node 3
 temp = number;       ----------  node 4
 while( temp != 0 ---------- node 5
 {
  remainder = temp/10; ----------node 6
  sum = sum + remainder*remainder*remainder;-----node 7
  temp = temp/10;  ----------node 8
  }
 if ( number == sum )  ----------node 9
 printf("Entered number is an armstrong number.");node 10
 else                 ----------node 11
 printf("Entered number is not an armstrong number."); node
12
return 0;        ----------node 13
}
```

In compiling this data with file size of 24.5KB no errors are traced.

Path selection and testing

Once the data is linked and compiled, control flow graph is to be constructed. Convert the given source code into graph that consists of nodes and edges.

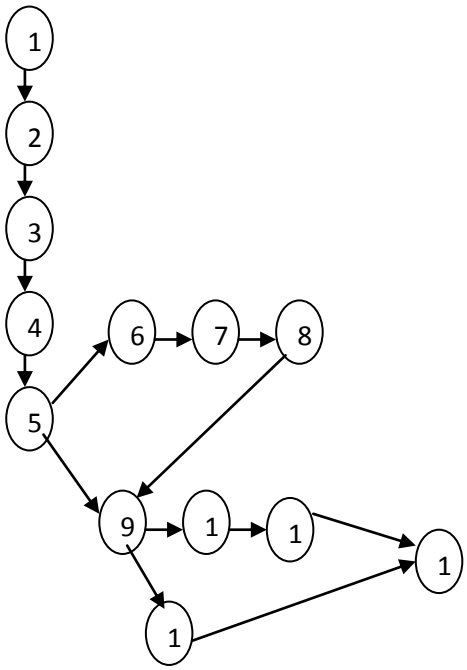

Figure 2 Initial control flow graph of the sample code

From the control flow graph compose sequencing nodes to form final control flow graph.

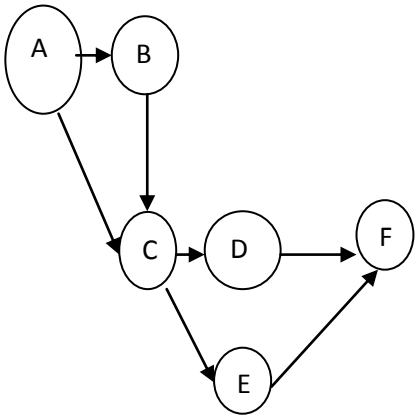

Figure 3 Final control flow graph of the sample code

Using the control flow graph all the paths present in the system are identified by using the following steps. Construct the adjacency matrix from the control flow graph.

Step 1: Designate any node as source node.

Step 2: From source node identify the immediate unvisited node and place it in a queue followed by the source node.

Step 3: Now repeat the same steps by considering the recently queued element as source node and repeat step 2.

Step 4: Continue the step 2 and step 3 until destination is reached.

Step 5: If destination is reached then go back to the source node to identify any addition paths if available.

The adjacency matrix for the above graph is

Nodes	A	B	C	D	E	F
A	0	1	1	0	0	0
B	0	0	1	0	0	0
C	0	0	0	1	1	0
D	0	0	0	0	0	1
E	0	0	0	0	0	1
F	0	0	0	0	0	0

Table 1 Adjacency matrix of the final graph

Step 1: Assume node A as the source node. Push it in the stack.

A

Recent adjacent node of A is A. From B recent adjacent node is C. Repeating the above procedure up to node F results in following path.

A B C D F

Pop one element from stack. (ie.F). Now from D check for any unvisited nodes.

Pop one element (ie.D). Now from C check for any unvisited nodes and push it into the stack.

A B C E F

Repeating the same procedure will result in identifying all the possible paths.

A C D F

A C E F

Once all the paths are identified then each path is tested to identify the faults.

Reliability estimation

Reliability specifies the fault free operation of a system. To estimate the system reliability it is essential to estimate the reliability of each path. From our examination the identified paths are

Path 1: A-B-C-D-F

Path 2: A-B-C-E-F

Path 3: A-C-D-F

Path 4: A-C-E-F

Reliability of path 1:

Total number of lines in path1= 5+3+1+2+1=12

Since there is an error in node 6(ie B), reliability of path1= 5/12=41.6%. This contributes to 30% of the entire system reliability. Since there is an error in this path 41.6% of 30% is 12.48%.All the other paths does not contain any error, hence their reliabilities are 100% each. Path 2, path 3 and path 4 contribute 27.5%, 22.5% and 20% respectively.

Hence system reliability = 12.48+27.5+22.5+20 = 82.48%

Hence in this module path reliabilities are estimated using which system reliability is computed.

Optimize results

Using the reliability results it is essential to optimize the solution since there is constraint of time and budget. In order to optimize the solution critical nodes and critical paths are identified. Critical node is a node that appears possibly in maximum number of paths. Critical path is the path that contains maximum number of critical nodes. In the specified example the critical nodes descending order are A, C, F, B, D, and E.

The critical paths are in the order

Path 1: A-B-C-D-F

Path 2: A-B-C-E-F

Path 3: A-C-D-F

Path 4: A-C-E-F

Hence the bug fixing has to be performed in the specified order because of the time constraint.

5. CONCLUSION AND FUTURE WORK

The analyses on the requirements are made. Design for the proposed system has been screened. The requirement analysis process includes learning and determining about the working environment, technical requirements and logical aspects or features of the system. The design of the system has been sketched based on the analyzed information.

The design has to be implemented in future, this design is applicable to certain changes as and when required in order to develop a prototype for the proposed work. The changes inserted would merely be in the physical components or other dependent components alone; the logical design of the system and its functionality would be preserved.

6. ACKNOWLEGEMENT

The authors would like to thank the anonymous reviewers for their valuable comments and suggestions to improve the presentation of this paper. The authors extend the sincere thanks to Rajalakshmi Engineering College for the constant support and encouragement.

7. REFERENCES

[1] S. S. Gokhale, "Architecture-based software reliability analysis: Overview and limitations," IEEE Trans.Dependable and Secure Computing, vol. 4, no. 1, pp. 32–40, Jan.–Mar. 2007.

[2] K. Go.seva-Popstojanova and K. S. Trivedi, "Architecture-based approach to reliability assessment of software systems," Performance Evaluation, vol. 45, no. 2/3, pp. 179–204, Jul. 2001.

[3] M.Xie, G.Y.Hong and C.Wohlin,"Software reliability prediction incorporating information from similar projects", Journal of software and systems,vol.49,No.1,pp.43-48, 1999.

[4] Y. K. Malaiya, M. N. Li, J. M. Bieman, and R. Karcich, "Software reliability growth with test coverage," IEEE Trans. Reliability, vol. 51, no. 4, pp. 420–426, Dec. 2002.

[5] C. Y. Huang and C. T. Lin, "Software reliability analysis by considering fault dependency and debugging time lag," IEEE Trans. Reliability, vol. 55, no. 3, pp.436–450, Sep. 2006.

[6] Roberto , S.Russo and K. S.Trivedi"Software reliability and testing time allocation-An architecture based approach" IEEE Trans. Reliability, vol. 36, no. 3, pp. 322–337, June. 2010.

[7] Lance fiondella and Swapna S.Gokhale,"Optimal allocation of testing effort considering the software architecture" IEEE Trans. Reliability, vol. 61, no. 2, pp. 580–589, June. 2012.

[8] Chao-Jung Hsu and Chin-Yu Huang,"An Adaptive Reliability Analysis Using Path Testing for Complex Component-Based Software Systems" IEEE Trans. Reliability, vol. 60, no. 1, pp. 158-170, March. 2011.