Operational profile based reliability assessment of COTS software

Tirthankar Gayen
IIT Kharagpur,
India

R. B Misra
IIT Kharagpur,
India

ABSTRACT
In this paper, approaches to assess the reliability of the COTS software for a given or specified operation profile have been analyzed using the proposed methods for *White box* and *Black box* approaches. In the black box approach the fragile point analysis is used to assess the reliability of the software, for a given operational profile, using the functional or design specification of the software.

General Terms
COTS: Commercial-off-the-shelf

CDG: Component dependency graph

CFG: Control flow graph

Keywords
COTS, CDG, CFG, software, reliability

1. INTRODUCTION

Most COTS software (*such as third-party libraries or executables*) rarely provides access to source code. For such a software component, with a given or specified operation profile, the reliability assessment can be made by two ways:

i) White box solution

ii) Black box solution

1.1 White box solution

In *white box solution* the black box component is converted to a white-box component using a suitable disassembler tool [4] [Gayen, Misra ,2008], which converts the binary executables or the object files to its equivalent assembly language code. From the assembly language, the CFG is generated, and from the CFG, the following evaluations are made.

Fault discovery evaluation

According to Norman Schneidewind [1] the expected number of faults at node ***n*** is given by

$$E(n) = p(n) * f(n) \quad (1)$$

where ***p(n)*** is the probability of traversing node ***n***,

determined by the branch probabilities

f (n) is the fault count in node ***n***

But this may not give the accurate results in all cases as there is no consideration of the probability of occurrence of the faults resulting in errors.

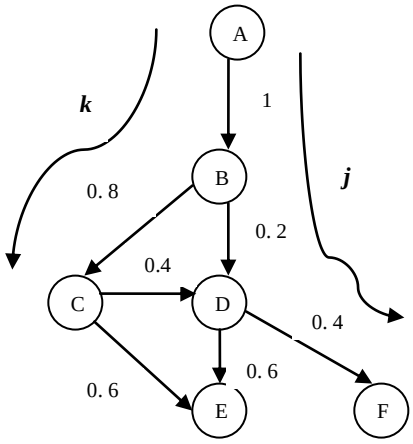

Fig 1. An example CDG with branch probabilities

From Fig. 1, the probability of traversing node ***B*** from ***A***
i.e ***p(B, A)*** =1
the probability of traversing node ***D*** from ***B*** i.e ***p(D,B)*** = 0.2
the probability of traversing node ***D*** from ***C*** i.e ***p(D,C)*** = 0.4
the probability of traversing node ***F*** from ***D*** i.e ***p(F, D)*** =0.4
and so on.
Hence, the probability of traversing node **B** i.e **p(B)**
= ***p(B, A***)= 1
the probability of traversing node **C** i.e **p(C)**
= **p(B)* p(C,B)** = 1*0.8 = 0.8
the probability of traversing node **D** i.e **p(D)**
= p(C)* p(D,C) + p(B)* p(D,B)
= 0.8*0.4 + 1*0.2 = 0.52

For a node ***n*** let there be ***k*** number of errors and let the probability of occurrence of error ***i*** *(causing failure)* be ***ci*** . Hence, the probability of non-occurrence of error ***i*** = ***1- ci***.

The probability of occurrence of errors ***1,2,3..,k*** are ***c1,c2, c3,*** … ***ck*** respectively for a particular operational profile. Hence, the

probability for non-occurrence of errors or the probability of success or the reliability of software given as

$$R(n) = (1- c1) * (1- c2) * \dots * (1- ck) \dots\dots k \text{ times}$$

The branch probabilities, can be easily determined from a given operation profile. The faults, causing errors at node ***n*** and their probability of occurrence is to be determined. The question is how to find the faults at node ***n*** with their probability of occurrence?

According to Sanyal et. al [2] the user can approximate the fault probability using the type of assembly level instructions and the number of such instructions that will be typically needed to manifest, the COTS function. For example, division and floating point instructions are more likely to failure, interrupt, and handlers depict transient characteristics and so on. This has been handled by [Gayen, Misra ,2009] [3,5] for the reliability assessment of COTS software.

1.2 Black box solution

In this process, a given black-box COTS component is used for a specific purpose, having some desired outputs. Consider the desired outputs as a set of ***n*** elements represented as ***{b1, b2, …, bn}***, where ***bi*** corresponds to a specific output ***i***. Based upon the functionality of the component, the input domain is divided into equivalent classes of subdomains and from each subdomain, test cases are selected. The boundary value analysis is also performed to select the test cases.

Let the test cases selected form a set of ***k*** inputs represented as ***{a1,a2, … , ak}.*** For a given or specified operation profile and the input domain, a survey of several runs of the component is taken into consideration with the test inputs ***{a1,a2, … , ak}*** selected from the input domain. From the survey made the results are noted.

Let the conditional probability that the output ***bk*** is produced, for the input ***aj*** is ***P(bk/aj)***

But, every output ***bk*** produced for input ***aj*** may not be correct, desired or acceptable. Therefore, an acceptance criteria ***cj*** is taken into consideration**.** When the output is correct or desired ***cj=1*** else ***cj=0.***

Hence, the probability that the correct output ***bk*** is produced is ***P(bk)*** for all inputs ***{a1,a2, … , ak}*** is

$$P(bk) = \sum_{j=1}^{J} P(bk/aj)*P(aj)*cj$$

Hence, the probability of obtaining the desired output or the reliability is

$$P(b1) + P(b2) + P(b3) + \dots + P(bn) = \sum_{i=1}^{n} P(bi)$$

The limitation of this process is that, it cannot be guaranteed that the test cases selected from the boundary value analysis and equivalent class partitioning cover all the execution scenarios or adequate enough to detect all the faults.

In this case the maximum coverage of all the execution scenarios can be obtained by keeping a record of all the inputs applied and the outputs obtained, for several runs of the program, for a given or specified operation profile.

This becomes feasible when the number of inputs or the input domain is small. But as the size of the input domain increases, the uncertainty in prediction also increases.

Consider software with ***n*** input variables. Variable V_1 may have k_1 number of values, V_2 may have k_2 and so on. Fig. 4 below shows all possible combinations of input variables $V_1, V_2, \dots V_n$ leading to outputs. The nodes in a particular column indicate all possible values that a variable V_i can have. The various paths from V_1 to V_n represents all possible combination of input variables the application can have resulting in output.

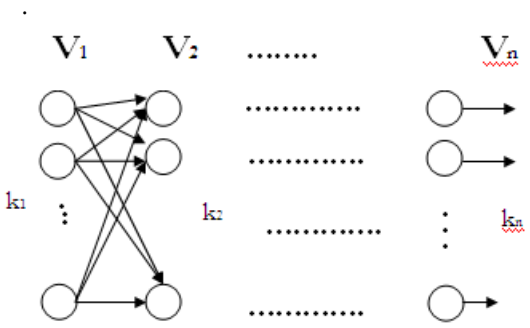

Fig. 4. The graph showing all possible combinations of input variables ($V_1, V_2, \dots V_n$) leading to outputs

For the software whose specification is available, the test cases can be reduced by performing the fragile point analysis.

2. FRAGILE POINT ANALYSIS

Fragile or weak point analysis is done in several areas like volcanic activity detection, where the possibility of the volcanic activity is detected by checking the fragile or weak areas of the earth's crust which are most susceptible or active for volcanic eruption. In an air tube, the worn out areas are the fragile or weak points susceptible for air leakage. In some cases some patch work may have been done in the worn out areas to handle the leakage if it occurs.

Similarly for software whose specification is available, the weak points are those weak or fragile areas of the specification which are susceptible to error.

For example, a software which is specified to evaluate the roots of a quadratic equation of the form $ax^2 + bx + c = 0$, with ***a, b*** and ***c*** as given inputs. The fragile points are:-

i) The input value of ***a***- If $a = 0$, then there can be a possibility of *divide by zero* error.

ii) If $b^2 < 4ac$ – There can be a possibility of *square root of a negative number* error.

iii) For $b >> 2a, (-b - \sqrt{b^2 - 4ac}) >> 2a, (-b + \sqrt{b^2 - 4ac}) >> 2a$ – There can be a possibility of *overflow* error due to divide operation.

iv) The values of $b^2, 4ac$ – There can be a possibility of overflow, if the values of b^2 or $4ac$ are large enough to exceed the maximum allowable value.

In some cases the fragile portions has been handled by the software itself (*similar to the patch work done in the worn out areas of the air tube*). For the above example, the software may have a built in feature to check whether a$=0$ for every input ***a*** and proceed accordingly. For example it may not accept the input ***a*** when it is equal to zero, generating appropriate messages asking to re-enter the value of ***a***. Hence, before performing any test activity it is advisable to detect the fragile points of the software. Once they are detected, appropriate test cases are used to detect the possibility of error at the fragile points. If an error occurs, then the test condition is noted, and all the input combinations/test cases which satisfies the test condition are eliminated from the test suite. From the operational profile, the probability of occurrence of all input combinations which satisfies the error condition is obtained. The reliability is evaluated using these values according to the proposed approach.

There may be several problems related to data inconsistency for a database management system. For example, in an online banking system, as per the design specification, for a joint account any one of the account holders should be allowed to login at a time for doing the transactions. Otherwise, there may result in data inconsistency. Consider a situation, where Rs 1000 is displayed as the account balance, to both the account holders of the joint account who wants to draw Rs 600. Since, the balance that is being displayed is sufficient to draw Rs 600, both the account holders places their option simultaneously. In such cases, there remain chances of data inconsistency, resulting in erroneous data, if such things are not handled properly. Therefore, the fragile point analysis checks these vulnerable points for the possibility of error.

3. THE PROPOSED APPROACH

For software with given specifications, the rules to predict the reliability of the software are as follows:-

i) From the specification of the software, detect the fragile points of the software.

ii) Appropriate test cases are used to detect the possibility of error at the fragile point.

iii) If an error is detected, the test condition is noted, and all the input combinations/test cases which satisfy the test condition are eliminated from the test suite.

iv) From the operational profile, the probability of occurrence of all input combinations $Pr(i,j)$, which satisfies the condition ***i*** for the fragile point ***j*** is obtained.

v) Step (iv) is repeated for all the conditions ***i***. for the fragile point ***j***

Mathematically,

$$Pr(j) = \bigcup_{i=1}^{m} Pr(i,j),$$

Where ***m*** is the number of conditions for fragile point ***j***

vi) Steps (ii) to (iv) are repeated for all the fragile points ***j***, and the probability of occurrence $Pr(j)$ obtained in step (v) is added at every iteration to obtain ***Q***.

Mathematically,

$$Q = \sum_{j=1}^{n} Pr(j)$$

Where ***n*** is the total number of fragile points.

vii) The probability of operational correctness R_{op} is obtained as $1-Q$

Mathematically, $R_{op} = 1-Q$

viii) From the remaining input space test cases are selected in accordance with the equivalent class partitioning and boundary value analysis.

ix) The test cases from step (viii) are applied as inputs the probability of logical correctness R_{log} is evaluated using Weiss and Weyuker 's model. [3,6]

x) The overall reliability ***R*** is the product of R_{op} and R_{log}.

Mathematically, $R = R_{op} * R_{log}$

Hence, using the black-box approach one can predict the reliability of a specification based software for a given operational profile. This approach is extremely useful as it

detects most of the errors including the operational errors. It also eliminates a whole of lot test cases which would otherwise have been required to test the software.

4. ILLUSTRATION

A software component which evaluates b!/5 - a2/ √(a-b) in the form of an executable file (i.e *'exp.exe*') is executed as shown in Fig. 5.

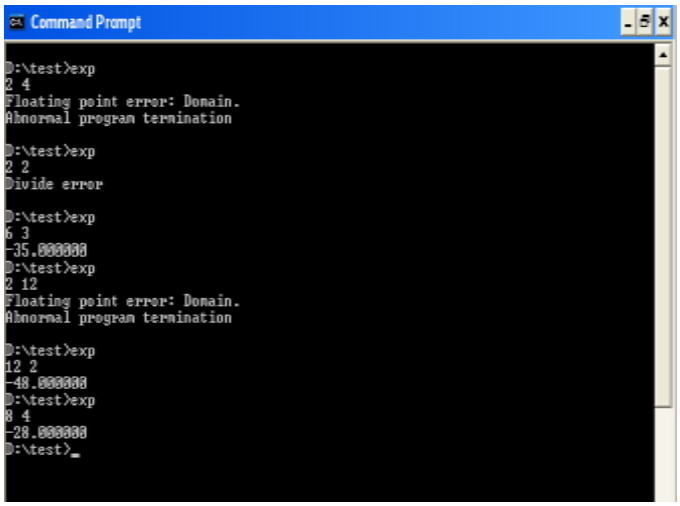

Fig. 5 The execution of '*exp.exe'*

Let us consider the input data to be a random value in the application domain. For 'a' let the specified domain be from -200 to 50000 and for 'b' let the specified domain be from 0 to 15. The possible errors are as follows:-

Out of range data set for 'a' = {[46341 …, 50000]}

The total number of integer data in this range = 3660

Probability of occurrence in this range =3660/50201=0.0729

Probability of non-occurrence in this range (executable range for 'a') =1- 0.0729 = 0.927

Out of range data set for 'b' = {[13 …, 15]}

The total number of integer data in this range = 3

Probability of occurrence in this range =3/16= 0.1875

Probability of non-occurrence in this range (executable range for 'b') =1-0.1875 = 0.8125

Divide by zero error occurs when a=b

Region of commonality between a and b is [0, … , 15]

Out of which the executable range is [0,…, 12]

The total number of integer data in this range = 13

The probability of occurrence of a=b is =1/13= 0.07692

The probability of non-occurrence of a=b = 1- 0.07692=0.923

Square root of a negative number occurs when a<b

The total number of integer data for a<b in the executable range = 12*11/2 + 13*200 = 2666

The probability of occurrence of a<b = 2666/(13*46541)= 0.0044

The probability of non-occurrence of a<b = 1- 0.0044 =0.99559

Logical error occurs when the output of a program does not match with the specified output is evaluated using Weiss and Weyuker et. al [3,6] 's model

Consider an example where the test suite consists of 4 test cases i.e

T = {(23, 12), (104, 4), (200, 10), (824, 7)}

Let a= 23, b=12 be a test case

The specified output should be 479001600/5 - 529/3 =95800143.667

The obtained output is = 95800144

|Difference| = 95800144 - 95800143.667 = 0.333

Let a=104, b = 4 be another test case

The specified output should be 24/5 - 10816/10 = -1076.8

The obtained output is = -1077

|Difference |= 1077 - 1076.8 = 0.2

Let a = 200, b = 10 be another test case

The specified output should be 3628800/5 - 40000/13=722683.0769

The obtained output is =725760 – 3076 = 722684

|Difference| = 722684 -722683.0769 = 0.9231

Let a = 824, b = 7 be another test case

The specified output should be 5040/5 - 678976/28 =23241.1428

The obtained output is =23241

|Difference| =23241.1428 – 23241 = 0. 1428

Considering the tolerance allowed i.e α = 0.9

Rlog = 1- 1/4{(0.333+0.2+0.9+0.1428)/ 0.9} =1- 0.437722= 0.562277

Overall reliability R= 0.927*0.8125*0.923*0.99559*0.562277 = 0.38916668

Thus, the reliability of the component is evaluated to be= 0.38916668

5. CONCLUSION

Using the fragile point analysis one can use the black-box approach to predict the reliability of a specification based

software for a given operational profile. The functional specification of the software can either be obtained from the vendor or from the design specification provided by the developer/software development company. This approach is extremely useful as it detects most of the errors including the operational errors. It also eliminates a whole of lot test cases which would otherwise have been required to test the software. Therefore, for a COTS component based software even if the source code unavailable, one can go for predicting the reliability of the software using this approach for the given functional or design specification of the software.

6. REFERENCES

[1] Norman Schneidewind, "Integrating testing with reliability", *Software Testing Verification and Reliability, Wiley* , Vol. 19, Issue 3, pp. 175-198, 2008.

[2] S.Sanyal, V.Shah, S Bhattacharya, "Framework of software Reliability Engineering Tool", *Proceedings of the 2nd High-Assurance Systems Engineering Workshop, IEEE Computer Society* , pp.114-119, 1997

[3] Tirthankar Gayen, R. B Misra , "Reliability assessment of Elementary COTS software component", *International Journal of Recent Trends in Engineering*, Issue 1, Vol 2, pp. 196-200, June 2009.

[4] Tirthankar Gayen, R. B Misra, "Reliability Bounds Prediction of the COTS Component Based Software Application", *International Journal of Computer Science and Network Security*, Vol 8, No. 12, pp. 219 – 228, Dec 2008.

[5] Vivek Goswami, Y.B.Acharya, "Method for Reliability Estimation of COTS Components based Software Systems" *in the proceedings of 20th International Symposium on Software Reliability Engineering, ISSRE 2009, IEEE Computer Society, IEEE Reliability Society, Computer Society of India,* Mysuru, India, Nov., 2009.

[6] S. N. Weiss, E. J. Weyuker, "An Extended Domain-Based Model of Software Reliability*", IEEE Transactions on Software Engineering*, Volume 14 , Issue 10, October 1988, pp: 1512 - 1524