INFORMATION PAPER
International Journal of Recent Trends in Engineering, Vol. 1, No. 2, May 2009

Reliability Assessment of Elementary COTS Software Component

Tirthankar Gayen[1], R.B Misra[2]

[1] Reliability Engineering Centre, IIT Kharagpur, India
Email: tgayen77@gmail.com
[2] Reliability Engineering Centre, IIT Kharagpur, India
Email: ravi@ee.iitkgp.ernet.in

***Abstract*— For predicting the reliability of a software application composed third party software components like COTS (*Commercial-Off-The-shelf)* one has to heavily rely on the reliability values available which may not be always correct. As, there can be several factors which can effect the reliability values for example, the vendors may be biased, it may not be possible to select test cases to execute all possible executable paths of the program (as the source code and other design specifications are not available). As, a result errors may still remain and can go undetected. The testing environment may not be the same as the operational environment, etc. Considering all these situations an innovative approach has been developed so that the integrators and the developers can make a relative comparison among the components to select suitable components to be integrated into the application.**

Index Terms— Software reliability, COTS, CDG, CFG

I. Introduction

Today, COTS (*Commercial-Off-The-Shelf*) play an increasingly important role in the software development. Due to financial and time-to-market considerations, the software development organizations have become increasingly reliant on software provided by third parties for functionality that is needed for the creation and maintenance of their applications. [1, 2] One of the most difficult problems for successful COTS component based system development is its reliability evaluation especially when the documentation and source code is not available.

According to ANSI, software reliability is defined as: *the probability of failure-free software operation for a specified period of time in a specified environment.* [7, 8] A software process is a logical process as opposed to physical process. Electronic and mechanical parts may wear out with time and usage, but software will not rust or wear-out during its life cycle. Software will never change over time unless intentionally changed or upgraded.

1.1 Software failure mechanisms

Software failures may be due to errors, ambiguities, oversights or misinterpretation of the specification that the software is supposed to satisfy, carelessness or incompetence in writing code, inadequate testing, incorrect or unexpected usage of the software or other unforeseen problems. [11] While it is tempting to draw an analogy between Software Reliability and Hardware Reliability, software and hardware have basic differences that make them different in failure mechanisms. Hardware faults are mostly *physical faults*, while software faults are *design faults*, which are harder to visualize, classify, detect, and correct. [7] Trying to achieve higher reliability by simply duplicating the same software modules will not work, because design faults can not be masked off by voting. The most important issue with the integration of the third party components like COTS components is its reliability prediction. According to Hoang Pham [12], in software, failures are caused by incorrect logic, incorrect statements or incorrect input data. Considering an idle operating environment for the components (i.e the operating environment is error free), the errors in software can be classified into two categories

* *Operational errors* – caused due to incorrect execution (For example: Out of range data, divide by zero, square root of a negative number)

* *Logical errors* – caused due to incorrect logic

These operational errors are not detected by the compiler because the compiler will give only the syntax and semantic error during the compilation of the code in accordance with the rules/grammar of the programming language. But, the operational errors like out of range data, divide by zero, and square root of a negative number occur only when appropriate error causing input data is given during the execution of the program. These errors are input data dependent and occur only during the execution of the program.

Errors due to requirements, design are all detected/captured during logical error detection. As there is no wear out failures in case of software (excluding the hardware dependencies). The third party software components like COTS (where the source code and other design information are hardly available) used for integration generally falls under the useful life. Since, no upgradation, debugging or maintenance is done during this phase. The software component whose reliability is

to be assessed lies in the *useful life with no upgrades*. Failures in software occur randomly. Hence, the only factor influencing failure is the input data used.

Therefore, the reliability of the component in this case should be expressed as a function from an input distribution or operational profile of the application. If a test set T is constructed by choosing inputs at random according to the distribution p (such as by using a random number generator), then T represents the operational profile, in theory, and may then be treated as a uniformly distributed set. [3]

Hence, the intention of this approach is to assess the reliability of the elementary COTS software component from the input distribution considering random inputs.

1.2 Assumptions:-

For a COTS component, it is assumed that the source code is not available. What is available is either binary object files (.OBJ) or binary executables (.EXE or .COM file in windows.) The binary programs are converted to equivalent assembly language programs using some disassembler tools like *Windows Disassembler*, *Bubble Chamber* which takes windows .exe or .com files as input to produce equivalent assembly language code. The *control flow graph*, CFG is drawn from the assembly language program [1, 2].

Also, no upgradation, debugging or maintenance is done and the component selected is of basic/elementary type performing a single function. The operating environment is considered to be idle (i.e free of operating environment errors).

II. THE PROPOSED APPROACH

The reliability of any component can be evaluated using Weiss and Weyuker et. al [3] 's model. It extends Nelson et. al [4]'s definition of the "probability of successful execution/reliability" R(P) of program P with respect to specification S is given as

$$R(P)= 1-\sum_{n \in N} p(n)*a(n)$$

where p(n) is the probability that input n is submitted to p

$$a(n) = 0 \text{ , if } P(n) = S(n)$$
$$=1 \text{ , otherwise}$$

If T is a test set T= { ti }, t ϵ T, $p_T(t)$ is the degree of representativeness to T; where

$p_T(t) = 0$ if $t \notin T$

If T is constructed by choosing inputs at random according to the distribution p (such as by using a random number generator), then T represents the operational profile, in theory, and may then be treated as a uniformly distributed set.
Therefore,

$$p_T(t) = 1/| T |$$

The generalized reliability

$$R = 1- 1/| T | (\sum_{t \in T} d_\alpha(Sp, P, t)/\alpha(t))$$

$d_\alpha (Sp, P, t)$represents α – discrepancy between Sp and P at t

α – discrepancy

For, arbitrary programs P and Q having the α – discrepancy between P and Q at n is

$d_\alpha (P, Q, n) = \min \{ \alpha(n), d((P(n), Q(n)) \}$,

$= d((P(n), Q(n)),$ if $P(n)\downarrow \cap Q(n)\downarrow$

$= 0,$ if $P(n)\uparrow \cap Q(n)\uparrow$

$= \alpha(n),$ otherwise

Where,

i) $d(x,y) > 0$ if $x \neq y$; $d(x,x) = 0$
ii) $d(x,y) = d(y,x)$, and
iii) $d(x,y) <= d(x,z) + d(z,y)$, where X is any set and x, y, z are elements of X

$P(n)\downarrow$ = P halts on input n

$P(n)\uparrow$ = P does not halt.

α(n) is the tolerance allowed at n

For any specification S with domain D and "don't care" set U and any program P

$Sp(n) = S(n)$, if $n \in D$
$= P(n)$, if $n \in U$

For any component the reliability assessment is done using the algorithm comprel which is as follows:-

Algorithm Comprel {

1) *Looking at the functionality of the component identify the possible operational errors for a given input domain of all the input parameters, the component may be subjected to during execution in a particular operating environment.*

The errors may be out of range data, divide by zero, square root of a negative number.

2) *Find out the probability of non-occurrence of each error for the given input domain of all the input parameters for the particular operating environment.*

3) *From the execution range of data design a test suite by selecting test cases based on the path coverage based testing of the component such that all the linearly independent paths of the CFG of the component (*obtained earlier) *are covered.*

In the path coverage-based testing strategy the test cases are selected from each decision point (shown in Fig. 1) such that all linearly independent paths in the program are executed at least once. A linearly independent path is any path through the CFG of the program that introduces at least one new node that is not included in any other linearly independent path.

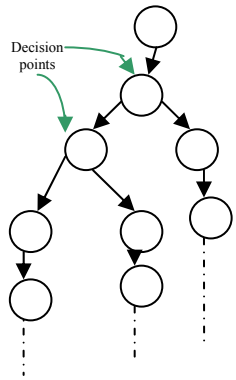


Fig. 1 A CFG with decision points.

McCabe's cyclomatic complexity which defines upper bound for the number of linearly independent paths can be used to determine the maximum number of linearly independent paths in CFG of the program.

The McCabe's cyclomatic complexity V(G) can be computed as:-

$$V(G) = E - N + 2$$

where N is the number of nodes in the CFG

E is the number of edges in the CFG

4) *Every test case t in the test suite* T *is executed and the output is compared with the specified output to obtain the α - discrepancy values.*

Mathematically,

$$d_\alpha (Sp,P,t) = \min \{ \alpha(t), d((P(t), Sp(t)) \},$$

if $P(n)\downarrow \cap Q(n)\downarrow$

$$= 0, \quad \text{if } P(t)\uparrow \cap Sp(t)\uparrow$$

$$= \alpha(t), \quad \text{otherwise}$$

5) *The logical reliability is obtained using the formula*

$$R_{log} = 1 - 1/|T| \left(\sum_{t \in T} d_\alpha (Sp, P, t)/\alpha(t) \right)$$

The, $\alpha(t)$ value is adjusted depending on the tolerance allowed at t.

6) *The overall reliability of the component is obtained by multiplying the probability of non-occurrence of the operational errors for all the parameters in their given input domain (*as obtained in step 2*) with the logical reliability value (*as obtained in step 5*).*

III. AN EXAMPLE APPLICATION

A software component which evaluates $b!/5 - a^2/\lfloor\sqrt{(a-b)}\rfloor$ in the form of an executable file (i.e *'exp.exe'*) is executed as shown below:-

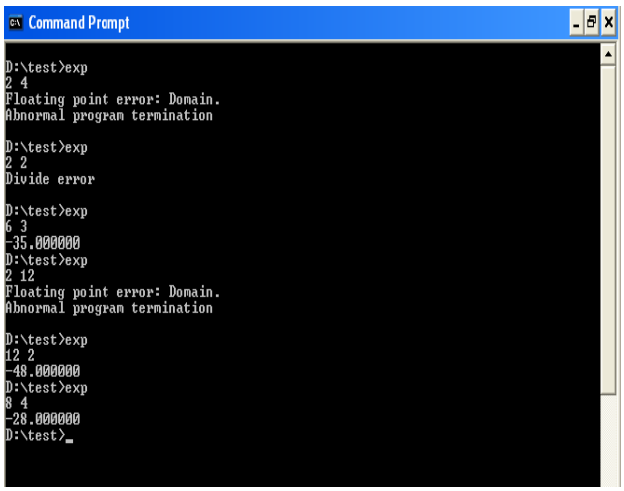
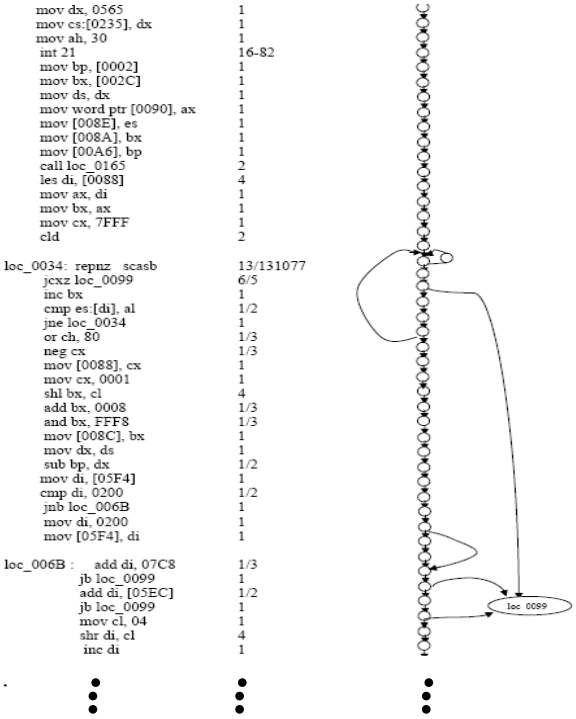


Fig. 2

The assembly language code (obtained using disassembler tools like *Windows Disassembler*) along with its CFG and CPU clock cycles are given as follows:

Code CPU clock cycles CFG



The ranges of integer values are:-

Short int -32768 to 32767 (16-bit)
Long int - 2147483648 to 2147483647 (32-bit)

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. Assuming 32-bit integer operation using 32-bit registers, the possible errors are as follows:-

Error – Out of range data

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

Error - Divide by zero

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

Error – Square root of a negative number

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

Error- Logical

Logical error occurs when the output of a program does not match with the specified output.
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 $\alpha = 0.9$

R_{log} = 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

IV. DISCUSSION AND CONCLUSION

This approach is an improvement over Weiss and Weyuker et. al [3] 's model. The problem of the Weiss and Weyuker et. al [3] 's model was in selecting the test cases from a particular input domain since there

was no guidelines regarding test case selection and occurrence of operational errors like out of range data, divide by zero, square root of a negative number. This problem has been solved by partitioning the input domain into operational error subdomain and logical subdomain, Test cases are selected according to the path coverage based tesing methodology and the reliability in the logical subdomain is predicted. This reliability value is multiplied with the probability of non-occurrence in the operational error subdomain to obtain the actual input domain based reliability value.

By using this approach the integrators and the developers can make a relative comparison among the components to select suitable components to be integrated into the application. But, to use this methodology for reliability assessment, one should be very careful in finding out all possible operational errors.

This approach/methodology could be of immense help to the integrators or developers to assess the reliability of the individual components before being integrated into the software application. Thus, using this approach they can easily take a rational decision, as whether to go for the development of application using the existing components or to discard it.

REFERENCES

[1] Tirthankar Gayen, R. B Misra, "Prediction of Upper Bound on the Reliability of COTS Component Based Software Application", *Proceedings of the International Conference on Quality and Reliability,* pp. 157-163, Chiang Mai, Thailand, Nov. 2007.

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

[3] 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, Pages: 1512 - 1524

[4] T. A Thayer, M. Lipow, and E.C Nelson, "Software Reliability" *(TRW Ser. Software Technol. 2)*, New York: North Holland, 1978.

[5] Shinji Inoue, Shigeru Yamada, "Generalized Discrete Software Reliability Modeling With Effect of Program Size", *IEEE Transactions on Systems, Man and Cybernetics, Part A,* March 2007,Volume: 37, Issue: 2, pages: 170-179.

[6] Peter G Bishop – "Rescaling reliability bounds for a new operational profile", *Proceedings of the 2002 ACM SIGSOFT International Symposium on Software Testing and Analysis*, Roma**,** Italy, 2002, pp. 180 – 190.

[7] Michael R. Lyu , *Handbook of Software Reliability Engineering.* McGraw-Hill publishing, 1995, ISBN 0-07-039400-8

[8] ANSI/IEEE, "*Standard Glossary of Software Engineering Terminology*", STD-729-1991, ANSI/IEEE, 1991

[9] Reliability Analysis Center, "Introduction to Software Reliability": *A state of the Art Review.* Reliability Analysis Center (RAC), 1996

[10] Keene, S. J., "Comparing Hardware and Software Reliability", *Reliability Review*, 14(4), December 1994, pp. 5-7, 21.

[11] Keiller, Peter A. and Miller, Douglas R., "On the Use and the Performance of Software Reliability Growth Models", *Software Reliability and Safety, Elsevier,* 1991, pp. 95-117.

[12] Hoang Pham – "System Software Reliability", *Springer series in reliability engineering* 2006, ISBN-10:18523339500.

[13] Jiantao Pan, "Software Reliability", *Carnegie Mellon University*, 18-849b *Dependable Embedded Systems* Spring 1999. http://www.ece.cmu.edu/~koopman/des_s99/sw_reliability

[14] Jean Dolbec and Terry Shepard, "A Component Based Software Reliability Model", *Proceedings of the 1995 conference of the Centre for Advanced Studies on Collaborative research*, p.19, November 07-09, 1995, Toronto, Ontario, Canada

[15] Sherif Yacoub, Bojan Cukic, and Hany H. Ammar, "A Scenario-Based Reliability Analysis Approach for Component-Based Software"- *IEEE Transactions on Reliability*, Vol. 53,No.4, 2004, pp. 465 – 480.

[16] William W. Everett, "Software Component Reliability Analysis", SPRE Inc.- *IEEE 1999*

[17] M.L. Shooman, "Software Engineering: Design, Reliability and Management", *McGraw Hill* 1983, ISBN 0-07-057021-3

[18] R. C. Cheung. "A User-Oriented Software Reliability Model." *IEEE Transactions On Software Engineering*, pp. 565-570, March 1980.

[19] Wen-Li Wang Ye Wu Mei-Hwa Chen- "An Architecture-Based Software Reliability Model"- *In Proceedings of the 1999 Pacific Rim International Symposium on Dependable Computing*, 16-17 December, pp. 143-150,Hong Kong, China. IEEE, 1999.

BIBLIOGRAPHICAL NOTES**:-**

Tirthankar Gayen, a B.E, M.Tech, Ph.D Scholar (IIT) working as a Senior Research Fellow at Reliability Engineering Centre, IIT Kharagpur. He has published papers in many international conferences and journals. His area of interest includes software reliability, neural network, natural language processing, etc.

email: tgayen77@gmail.com

R. B Misra, a B.E, M.Tech, Ph.D, Professor, Reliability Engineering Centre, IIT Kharagpur, he is a Senior Member of IEEE, and Fellow at Institution of Engineers (India). He has published papers in many papers in international conferences and journals. His area of interest includes power system reliability, reliability design and testing, software reliability, system safety, etc.

email: ravi@ee.iitkgp.ernet.in