Reliability Bounds Prediction of COTS Component Based Software Application

Tirthankar Gayen[†] **and R. B Misra**[††],

Reliability Engineering Centre
IIT Kharagpur-721302, India.

Summary
In this paper a unique methodology based on the execution scenario analysis of the COTS component based software application has been formulated to help the developers and integrators to regain some control over their COTS component based software application systems by predicting the upper and lower bound on the reliability of their application systems. At the component level the CFG (*control flow graph*) of the component and at the application level the CDG (*component dependency graph*) of the application are explored to identify all possible execution scenarios. The maximum and minimum reliability values are obtained by evaluating and comparing the reliability values obtained from various execution scenarios.

Key words:
Software, Reliability, CDG, CFG, COTS

1. Introduction

The use of third party software components, such as **COTS** (*commercial-off-the-shelf*) products, has become more and more common in the building and maintenance of large software systems. Corporate downsizing decreased government budgets, minimum spiraling costs of building and maintaining large software systems, have necessitated the reuse of existing software components which can potentially reduce the time-to-market. Programmers are expensive. Instead of paying an entire team of developers to create a component from the ground up it is cheaper to simply pay a few developers to integrate a pre-existing component into a new application. That is why today very few large-scale software systems are built from the scratch. They generally comprise of commercial-off-the-shelf (COTS) components, legacy software, and custom-built components.

COTS components are defined by Vigder and Dean as "*components which are bought from a third-party vendor and integrated into a system*" [1]. A COTS component could be as "small" as a routine that computes the square root of a number or as "large" as an entire library of functions created by people outside of the software development organization that will actually use it. Though employing COTS components in the building and maintenance of a large system can provide some benefits, yet COTS component usage presents some unique problems as follows [21] :-

1. COTS component source code is often unavailable.
2. Updates and evolution of a COTS component are provided by the vendor. New functionality of an updated component could be detrimental to specific applications that use it. In fact, the functionality in the original component could also be problematic.
3. The vendor often fails to provide a correct or complete description of the COTS component's behavior. This can result in the buyer of the component having to guess how the component is meant to be used or how it is supposed to behave. Worse yet, the buyer could end up using the component in a manner the vendor did not intend. Unanticipated uses could compromise the reliability of both the COTS component and the application into which it is integrated.
4. Maintenance can become an issue because the vendor may not correct defects or add enhancements as the buyer needs them. Developers in the organization that purchased the component may be forced to make modifications themselves, which can be quite difficult if the component's source code is unavailable or if the component's specification is poor. On the contrary, integrating COTS components into an application is prone to error, can require a significant amount of coding, and can be problematic to test properly.

The reliability of components affects the reliability of the system. Since today COTS (*Commercials-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. One of the most difficult problems for successful COTS component based system development is its evaluation especially

Manuscript received December 5, 2008
Manuscript revised December 20, 2008

when the documentation and source code is not available. There are several questions which arise like:

i) How to estimate the reliability of COTS when there is no data available from the vendor?
ii) How to estimate the reliability of COTS when it is embedded in a larger system?
iii) How to revise the reliability estimates once COTS has been upgraded?

Therefore, it is the intention to develop a methodology to help the developers and integrators to regain control over their COTS component based software application systems by predicting the upper and lower bound on the reliability of their application systems.

2. Problem Definition

Sherif et. al's [28] approach for Scenario-Based Reliability Analysis considers the transition probability whose accurate evaluation may not be very easy since the transition from one component to another may depend on several factors like user input data or other conditions which may depend on several instance characteristics. Dolbec et. al's model for Component Based Software Reliability demonstrates that the Shooman's execution path model can be transformed into a component based model. Their model defines the reliability of a system as a function of the reliability of the components and components usage ratios. The new definition of system unreliability is derived as

$Q_s \approx \phi_1 D_1 + \phi_2 D_2 + \ldots + \phi_m D_m$.

where m represents the number of components that are used during system execution

ϕ_m represents the usage ratio of component m in N tests

D_m represents the probability of failure of component m.

or

$$Q_s \approx \sum_{k=1}^{m} \phi_k D_k \qquad (1.1)$$

Equation 1.1 can also be expressed as follows:-

$$Q_s \approx \sum_{k=1}^{m} \phi_k (1 - C_k) \qquad (1.2)$$

where C_k represents the component reliabilities.

Software system reliability, R_s is equal to:

$$R_s = 1 - Q_s \approx 1 - \sum_{k=1}^{m} \phi_k D_k \qquad (1.3)$$

Component Usage Ratio – It represents the ratio of total component execution time over the total software system execution time. The value of the component usage ratio is $0 < \phi_k < 1$. The total of all components usage ratios is equal to 1.

$$\sum_{k=1}^{m} \phi_k = 1$$

$\phi_k = t_k/t_s =$ total component execution time(t_k)/ total system execution time (t_s)

Since the execution time is machine dependent and will vary with varying system load; determining component execution time is more difficult because it is necessary to track when each component is executed and for how long. Moreover, it is execution path independent, loops and other execution instance characteristics are not taken into consideration. Therefore, considering all these situations the current aim of the research is to formulate a methodology to eradicate these problems and to help the developers and integrators to regain some control over their COTS component based software application systems by being able to predict the reliability of their application systems.

3. Upper bound prediction

In this approach before designing a component based software application, the developer evaluates the reliability of the individual component like COTS (especially when the source code is not available) obtained from the third party using John D. McGregor et. al' s [26] method for Measuring Component Reliability. At first the ***inter-component analysis*** is done by drawing the CDG (*Component Dependency Graph*) for the given component based software application.

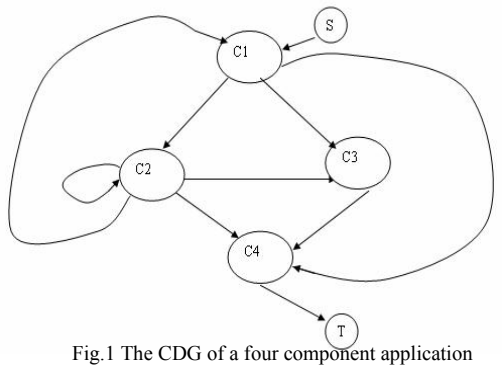


Fig.1 The CDG of a four component application

Here C1, C2, C3 and C4 represent components. The loops in CDG represent the repeated execution of the component (*like during a recursive call*) (C2 has a loop in fig 1.) and a cycle represents the repeated execution of the sequence

of components included within the cycle. (*For example a loop within the application.*) (C1 - C2 - C1 represents a cycle in Fig. 1.) From the CDG the possible execution scenarios are separated out as follows:-

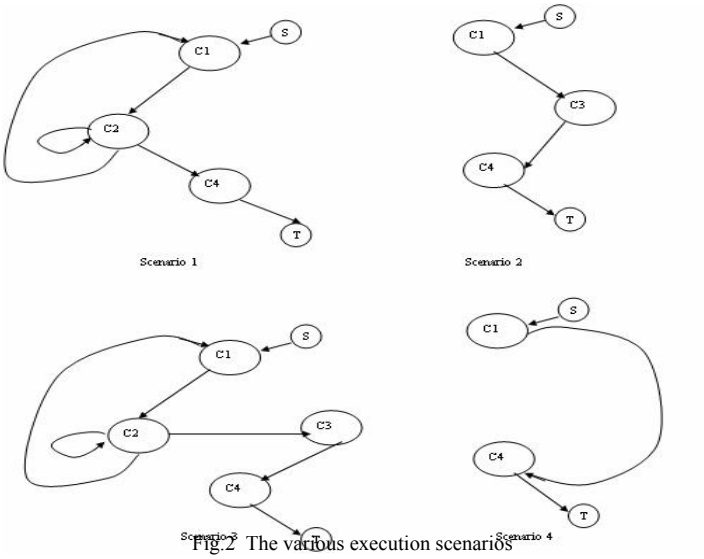


Fig.2 The various execution scenarios

Using Dolbec et. al 's model for Component Based Software Reliability the *Component Usage Ratio* (which represents the ratio of total component execution time over the total software system execution time) is evaluated.

$\phi_k = t_k/t_s$ = total component execution time(t_k)/total system execution time (ts)

where the value of the component usage ratio is $0 < \phi_k < 1$ and total of all components usage ratios is equal to 1. i.e

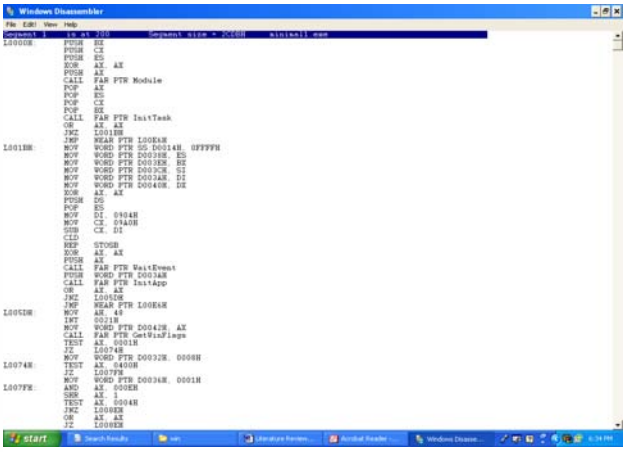
$$\sum_{k=1}^{m} \phi_k = 1$$

Since the execution time of a program is machine dependent and will vary with varying system load. Hence, it is resorted to calculate the execution time in terms of the number of CPU clock cycles as the execution time is directly proportional to the number of CPU clock cycles.

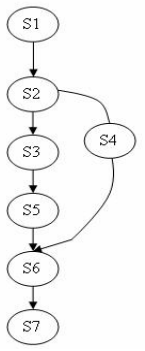
For this the ***intra - component analysis*** is done. Consider a COTS component, assuming that the source code is not available. What is available is either binary object files (.OBJ) or binary executables (.EXE or .COM file in windows.) The binary programs are converted to equivalent assembly language 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. Below is an example of a sample Windows executable (.EXE) file during execution.

Press.exe is a simple windows binary executable file which when executed displays a window with the message "Press Me!" and finally closes the window on mouse click.

Corresponding assembly language code generated by *Windows Dissassembler* for *Press.exe*



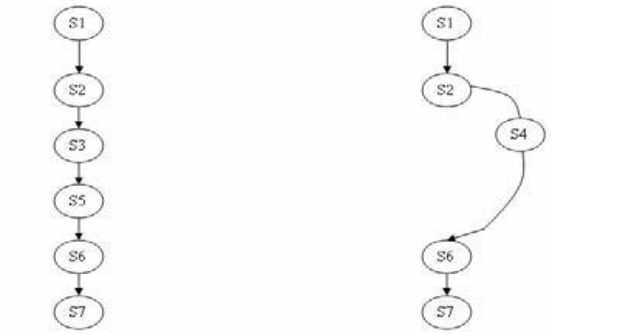
The *control flow graph*, CFG is drawn from the assembly language program. Then the CFG is analyzed to find all possible execution scenarios. Consider the example



An example CFG

Here S1, S2, S3, S4, S5, S6, S7 represents statements and the arrows represents the flow of control.

The above CFG can be broken into two execution scenarios as follows:-



Scenario 1 Scenario 2

The definite loops and definite cycles are numbered to represent the number of repetitions. Here the statements S1, S2, S3, S4, S5, S6 and S7 may have different execution times. Referring to the machine instruction set manual according to the assembly language specifications one can evaluate the number of CPU clock cycles required to execute various instructions. Thereby evaluating the number of CPU clock cycles needed to execute the program. Again, there can be a problem with the loops which can be definite, indefinite or conditional. With definite loops the evaluation is easy but not with indefinite conditional loops. All these problems are handled in the following algorithm which directly gives the upper bound of reliability of a COTS component based software application.

Algorithm GayenCOTS()

{

i) Do the intra-component analysis to find those components which have indefinite loops from their CFG.

ii) Do the inter-component analysis to find those sequences of components which are in indefinite cycles from their CDG.

iii) Among all the components found from step (i) & (ii) during intra and inter-component analysis respectively find the components having maximum reliability.

iv) If all the components either have indefinite loops or are in indefinite cycles then the upper bound is the reliability of the component having maximum reliability, hence terminate.

Else {

v) Compare the reliability value of the components obtained in step (iii) with the reliability values of other components in the application not found in steps (i) & (ii).

vi)If the reliability of the component found in step (iii) is greater after comparison in step (v) then one can immediately conclude that the upper bound on reliability of the application is the reliability of the component found in step (iii), hence terminate.

The proof is as follows:-

According to Dolbec et. al's model for component-based software reliability the component usage ratio

ΦC_i = total component execution time /total system execution time

$= T(C_i) / T(S_k)$

The reliability of the execution scenario k is

$R(S_k) = \sum \Phi C_i * R(C_i)$

$\forall C_i \in S_k$

where $R(C_i)$ is the reliability of the component C_i, S_k corresponds to the execution scenario k, $R(S_k)$ is the reliability of the execution scenario k

or

$R(S_k) = \sum (T(C_i) / T(S_k) * R(C_i))$

$\forall C_i \in S_k$

or it can be written as

$R(S_k) = T(C_1) / T(S_k) * R(C_1) + T(C_2) / T(S_k) * R(C_2) + ..$

where $C_1, C_2, \ldots \in S_k$

It is known that the component usage ratio $T(C_i) / T(S_k)$ of a component increases with the increased use of a component C_i .Therefore, from the equation

$R(S_k) = T(C_1) / T(S_k) * R(C_1) + T(C_2) / T(S_k) * R(C_2) +....$

it can be said that if the component C_i takes infinitely huge amount of time to execute or is used for infinitely large number of times, so that, under limiting situation, the component usage ratio $T(C_i) / T(S_k)$ for the component C_i tends to 1 and the component usage ratio of all other components tends to zero. Hence, the conclusion drawn in steps (vi) is proved.

vii) Find out the minimum execution time for the components having indefinite loops or in indefinite cycles found in steps (i) & (ii).

The evaluation of minimum execution time for the components having indefinite loops or in indefinite cycles

For an indefinite conditional loop, if the condition checking is done at the beginning of the loop the instructions inside the loop are skipped and the next statement after the loop ends is considered. This is because of the interest in evaluating the minimum execution time of the components having indefinite loops. It is also because of the same reason that the user interface and

other delays are neglected. For a conditional loop, if the condition checking is done at the end of the loop then the instructions inside the loop is executed at least once (or the loop is executed at least once). It is because at least one pass through the loop will take place. Hence, executing the sequence of instructions in the loop only once will serve the purpose. Since more number of passes through the loop will increase the execution time of the component and as a result will subsequently increase the component usage ratio.

The execution times are added according to the sequence of statements in the execution scenarios to obtain the total execution time of each scenario of the CFG for a particular component. The execution times of all the scenarios for a particular component are compared to obtain the minimum execution time.

viii) Find the execution times for other components in various execution scenarios of the CFG of the component. (not having indefinite loops or in indefinite cycles).

This is done by adding the execution times of all the sequence of statements in the execution scenarios to obtain the total execution time of each scenario of the CFG for a particular component.

Therefore, mathematically the total execution time of scenario k for a component j is

$T_j(CFGS_k) = \sum_{S_i \in CFGS_k} T(S_i)$,

where $T(S_i)$ represents the execution time of statement-Si
$CFGS_k$ represents execution scenario k
$T_j(CFGS_k)$ represents the time to execute the execution scenario k for component j.

ix) Do inter-component analysis by considering various execution scenarios of the CDG (Component Dependency Graph) of the component based application to calculate the execution time of each execution scenario by adding the execution time of each component present in the execution scenario with the component interfacing time in terms of the number of CPU clock cycles (which is readily available to the developer considering the compiler specifications of his source code).

From the various execution scenarios of the CDG (Component Dependency Graph) of the application the execution time to execute the execution scenario k is

$$T(S_k) = \sum_{\forall CFGS_m \in S_k} T_i(CFGS_m) + T(interface),$$

where S_k represents the execution scenario k,
$T(S_k)$ represents the time to execute the execution scenario k,
T(interface) represents the interfacing time,
$T_i(CFGSm)$ represents the time to execute the execution scenario m of the CFG of component C_i.

x) By varying the values of $T_i(CFGSm)$ for various execution scenarios of the CFG of all components $C_i \in S_k$ with its corresponding value of $T(S_k)$ the different reliability values $R(S_k)$ for a particular execution scenario k of the CDG are obtained using the formula below.

The reliability of the execution scenario k is evaluated using the formula

$$R(S_k) = \sum_{\forall C_i \in S_k} \Phi C_i * R(C_i)$$

where $R(C_i)$ is the reliability of the component C_i,
S_k corresponds to the execution scenario k of the application,
$R(S_k)$ is the reliability of the execution scenario k of the application

or

$$R(S_k) = \sum_{\forall C_i \in S_k} T_i(CFGS_m) / T(S_k) * R(C_i)) + T(interface)/T(S_k)$$

where $R(C_i)$ is the reliability of the component C_i ,
$T_j(CFGS_m)$ represents the time to execute the execution scenario m of the CFG of component C_i,
$T(S_k)$ represents the time to execute the execution scenario k.
T(interface) represents the interfacing time.

(*Assuming that the system developer does correct interfacing of the components. It can be considered that the interfacing part as a component having reliability equal to 1.*)

xi) In this way obtain multiple reliability values for all the scenarios of the CDG of the application.

xii) By comparing the values of the reliabilities of all the scenarios of the CDG of the application obtain the maximum value which corresponds to the upper bound on the reliability of the application.

}

}

Thus, the estimate of the upper bound on the reliability of the COTS component based application is evaluated.

4. Comparison of Dolbec's method with algorithm GayenCOTS

Consider an example in which Dolbec's estimation did not give proper results.

The Component Dependency Graph (*CDG)* of the software application is as follows:-

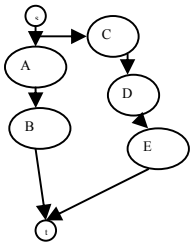
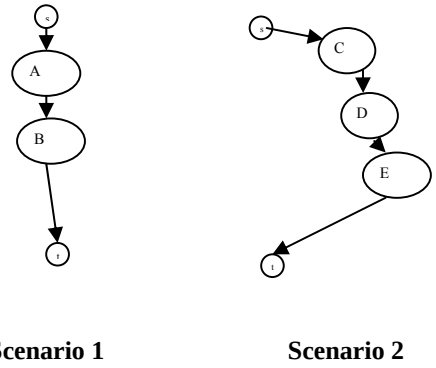


Fig. 3

The reliability values of the components are available to us using John D. McGregor et. al' s [26] method for Measuring Component Reliability Referring to the machine instruction set manual according to the assembly language specifications the number of CPU clock cycles required to execute various instructions .are evaluated. The maximum and minimum execution time of the individual components are obtained during intra-component analysis (described in Gayen, Misra 2007)[34].

S.No	Component	Reliability value	Max. time	Min. time
1.	A	0.99	453	104
2.	B	0.94	385	102
3.	C	0.88	107	22
4.	D	0.81	1128	516
5	E	0.83	53	17

The various execution scenarios corresponding to the CDG *(Fig.3*) are as follows:-



Scenario 1 **Scenario 2**

Fig. 4 The various execution scenarios

From scenario1 the reliability value = 0.9306
From scenario 2 the reliability value = 0.591624

Therefore, the maximum reliability value possible is 0.9306

According, to algorithm GayenCOTS the upper bound for scenario1
= (0.99*453 + 0.94*385)/838 = 0.9670
According, to algorithm GayenCOTS the upper bound for scenario2
= (0.88*107 + 0.81 * 1128 + 0.83 * 53)/1288
= 0.8166
Hence, according to GayenCOTS the upper bound of the application is 0.9670

According, to Dolbec the upper bound is
= (0.99*453 + 0.94*385 + 0.88* 107 + 0.81*1128 + 0.83*53)/2126
= 0.8759

Hence, it is seen that the Dolbec's estimation gave a lower value (i.e 0.8759) of upper bound than the original maximum reliability value possible (i.e 0.9306) in this case.

Whereas, the algorithm GayenCOTS gave a higher value(i.e 0.9670) (close to the original value) of upper bound than the original maximum reliability value possible (i.e 0.9306) in this case.

Therefore, algorithm only GayenCOTS gave the correct value of upper bound in this case.

Hence, it is verified that under all situations algorithm GayenCOTS is better than that of Dolbec. It is because algorithm GayenCOTS will give upper bound under all situations but Dolbec's method would fail (as in this case).

5. Lower bound prediction

For a given COTS component based software application system the lower bound can be evaluated by:-

* Dividing the system into various execution scenarios and evaluating separately the minimum reliability of each execution scenario.
* Then comparing the minimum reliabilities obtained from all the execution scenarios to get the minimum.

But the problem is in evaluating the reliability of the components which are either in indefinite loops or in indefinite cycles.

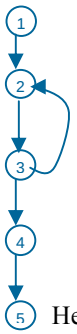
The problem can be handled in the following ways:-

1) If the application is time bound then the maximum possible number of repetitions through the loop or cycles can be found by considering the execution time of the application.
2) In reality, all the applications may not be time bound, they may be user dependent and may continue to execute for an indefinite period of time. For example the Web server.

But in accordance with the definition of reliability *"Reliability is defined to be the probability that a component or a system will perform a required function for a given period of time when used under stated operating conditions."*

Hence, it is evident that reliability is time bound. Or in other words even if the application may execute for an indefinite period of time, the reliability value is evaluated only for a specified *given period of time*.

Consider an execution scenario containing a single loop



The given specified period of time for the application is T_{appl}.

Let n be the maximum number of possible repetitions through the loop.

The value of n is calculated as

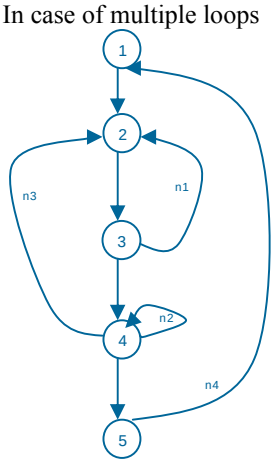
$n = \text{floor}((T_{appl} - (T_1 + T_4 + T_5))/(T_2 + T_3))$

where T_1, T_2, T_3, T_4, T_5 are the execution times of component 1,2,3,4,5 respectively and floor(x) = largest integer less than or equal to x

Here, the interfacing time of the component in included in the component execution time. Also, the user interface delays and other delays are neglected as only the maximum number of possible repetitions through the loop is taken into consideration. Once the value of n is obtained then the reliability is predicted using the formula

$R = R_1*(R_2*R_3)^n*R_4*R_5$ which will be the minimum reliability for this execution scenario.

In case of multiple loops



Evaluate the maximum number of repetitions possible in each loop n which in this case are n1, n2, n3, n4 for the corresponding loops as shown in the diagram. The value of n1 can be evaluated by assuming that all the components other than what is enclosed in the loop is executed only once.

Therefore,

$n1 = \text{floor}((T_{appl} - (T_1 + T_4 + T_5))/(T_2 + T_3))$

where, T_1, T_2, T_3, T_4, T_5 are the execution times of component 1,2,3,4,5 respectively.

Assuming an ideal developer with perfect interfacing capabilities the interfacing reliability is assumed to be 1.

Similarly

$n2 = \text{floor}((T_{appl} - (T_1 + T_2 + T_3 + T_5))/T_4)$

$n3 = \text{floor}((T_{appl} - (T_1 + T_5))/ (T_2 + T_3 + T_4))$

$n4 = \text{floor}((T_{appl}) /(T_1 + T_2 + T_3 + T_4 + T_5))$

where, T_1, T_2, T_3, T_4, T_5 are the execution times of component 1,2,3,4,5 respectively and

floor(x) = largest integer less than or equal to x

The reliability value obtained considering maximum repetition of loop 1 is $R_{n1} = R_1*(R_2*R_3)^{n1}*R_4*R_5$

The reliability value obtained considering maximum repetition of loop 2 is $R_{n2} = R_1*R_2*R_3*(R_4)^{n2}*R_5$

The reliability value obtained considering maximum repetition of loop 3 is $R_{n3} = R_1*(R_2*R_3*R_4)^{n3}*R_5$

The reliability value obtained considering maximum repetition of loop 4 is $R_{n4} = (R_1*R_2*R_3*R_4*R_5)^{n4}$

The minimum reliability value is obtained by comparing the reliability values of obtained i.e R_{n1}, R_{n2}, R_{n3}, R_{n4} to obtain the minimum.

or

$R_{min} = \min (R_{n1}, R_{n2}, R_{n3}, R_{n4})$

This value gives the minimum possible reliability value for this execution scenario.

In a similar manner the reliability values of the other execution scenarios are obtained.

The lower bound on the reliability of the application is obtained by comparing the minimum reliability values of various execution scenarios to obtain the minimum.

or

$R_{lower} = \min(R_{S1}, R_{S2}, R_{S3}, R_{S4})$

This gives the lower bound on the reliability of COTS component based software application.

6. Implementation of the approaches on a real application

A software which evaluates b!/5 - a^2/ √(a-b) in the form of an executable file (i.e *'exp.exe'*) is executed as shown below:-

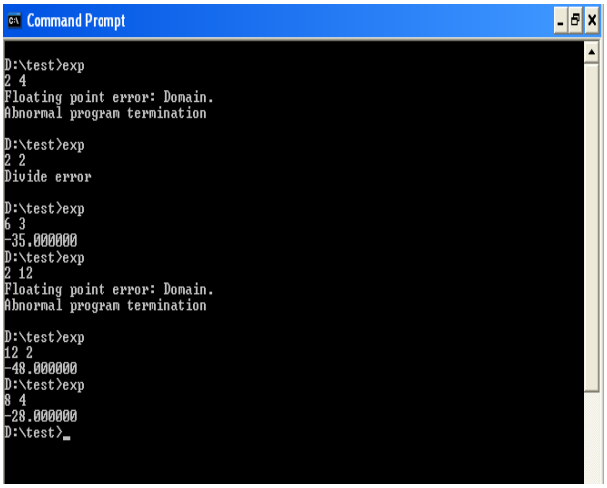
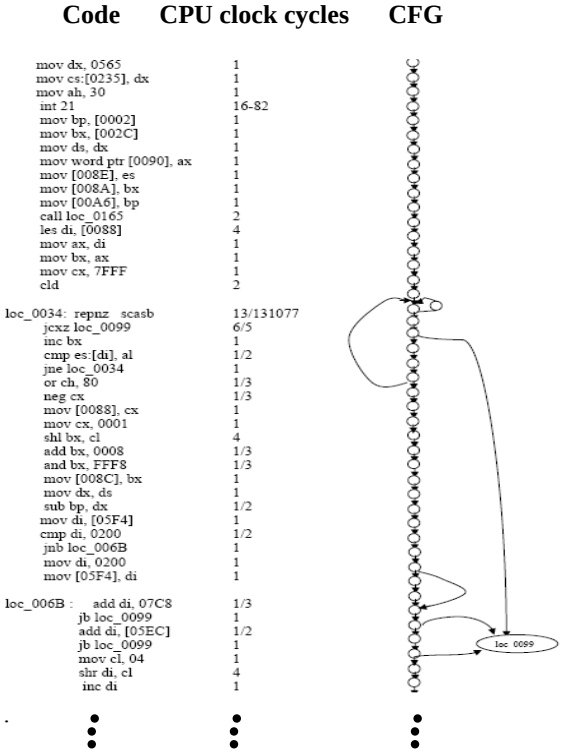


Fig. 5

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



Let us consider this example where the reliability values of the components are available to us using John D. McGregor et. al' s [26] method for Measuring Component Reliability Referring to the machine instruction set manual according to the assembly language specifications the number of CPU clock cycles required to execute various instructions .are evaluated. The maximum and minimum execution time of the individual components are obtained during intra-component analysis (described in Gayen, Misra 2007)[34].

According to the algorithm GayenCOTS the Upper Bound for scenario1
= (0.99*453 + 0.92*107 + 0.82*1128 + 0.98*53)/1741
= 0.875249856

According to the algorithm GayenCOTS the Upper Bound for scenario2
= (0.99*453 + 0.92*107 + 0.82*1128)/1688
= 0.87196

Hence, the upper bound according to GayenCOTS is 0.875249856

According to Dolbec the Upper Bound

= (0.99*453 + 0.86*385 + 0.92*107 + 0.82*1128 + 0.98*53)/2126
= 0.872488

The Lower Bound for scenario1
= 0.731918

The Lower Bound for scenario 2
= 0.7468

Hence, the lower bound of the application is 0.731918

Therefore, the reliability bound of the application 0.731918 to 0.875249856.

7. Conclusion

The increasing use of third-party COTS components can in theory lead to reduced costs and faster development cycles, but these advantages can come at a steep price. With their applications dependent on the behavior of components from third parties, developers and integrators have suffered a loss of control. Thus, the goal has been to help developers and integrators regain some control of their COTS-based software application systems by predicting the upper and lower bound on the reliability on their COTS-based software application systems. As by knowing the upper and lower bound, one can easily predict the range of reliability values an application can have.

The algorithm for the prediction of upper bound was an improvement over Dolbec et. al's [27] model for Component Based Software Reliability. The drawback of his model was that it was execution path independent and component interfacing time is not taken into consideration. Therefore, it was unable to predict the upper bound on reliability, as the upper bound on reliability obtained using Dolbec et. al 's [27] model for the example considered is much less than the value obtained in the proposed approach. It is valid under any processing environment be it batch or parallel processing in a uniprocessor or a multiprocessor system. Since, here one is mainly concerned with the component usage ratio and not with the clock time for the execution of the component. Here, it is assumed that the developer is an ideal developer who codes correctly (as the reliability of the interfacing code is considered to be unity) interfacing the COTS components without any error (which may later on cause failure of the application system) to produce the product which may not always be in reality.

The lower bound on the reliability of the application was predicted using the scenario based reliability analysis gives the minimum reliability the application can have. The approach is valid only for sequential processing system. Since, today parallel processing is in vogue. Efforts are in progress to incorporate this feature so that it becomes more versatile.

With this research, an innovative approach has been developed to predict the upper and lower bound on the reliability of the COTS component based software application. A unique methodology based on the execution scenario analysis of the COTS component based software application has been formulated.

References

[1] M. Vigder, J. Dean, "An architectural approach to building systems from COTS software components," *Technical Report* 4022 1, National Research Council, 1997.

[2] Allen, Robert and David Garlan. A Formal Basis for Architectural Connection, *ACM Transactions on Software Engineering and Methodology,* 1997.

[3] Cho, Il-Hyung and McGregor, John D. "Component Specification and Testing Interoperation of Components", IASTED 3rd *International Conference on Software Engineering and Applications*, Oct.1999.

[4] Hissam, Scott, Gabriel A. Moreno, Judith Staffod, Kurt C.Wallnau. "Packaging Predictable Assembly with Prediction-Enabled Component Technology," *Computer Society*-1071-9458/05-2005.

[5] Mason, D. "Probabilistic Analysis for Component Reliability Composition," Proceedings of the 5th ICSE *Workshop on Component-Based Software Engineering,* Orlando, Florida, May 2002).

[6] Musa, John. Software Reliability Engineering, New York, NY, McGraw-Hill, 1998.

[7] Stafford, Judith A. and McGregor, John D., "Issues in Predicting the Reliability of Components," Proceedings of the 5th ICSE *Workshop on Component-Based Software Engineering*, Orlando, Florida, May 2002.

[8] Szyperski, Clemens. *Component Software:Beyond Object-Oriented Programming,* Addison-Wesley, 1998.

[9] S. Gokhale et al., "Reliability simulation of component-based software systems," in Proc. 9th *Int. Symp. Software Reliability Engineering* (ISSRE'98), Paderborn, Germany, 1998, pp. 192–201.

[10] S. Krishnamurthy and A. P. Mathur, "On the estimation of reliability of a software system using reliabilities of its components," in Proc. 8th *Int. Symp. Software Reliability Engineering* (ISSRE'97), Albuquerque, New Mexico, Nov. 1997, pp. 146–155.

[11] D. Mason and D. Woit, "Problems with software reliability composition," in 9th Int. *Symp. Software*

Reliability Engineering (ISSRE'98), Paderborn, Germany, 1998, Fast Abstracts, pp. 41–42.

[12] B. Meyer et al., "Building trusted components to the industry," *IEEE Comput.*, pp. 104–105, May 1998.

[13] J. Voas, "Error propagation analysis for COTS systems," *IEEE Comput*. Control Eng. J., vol. 8, no. 6, pp. 269–272, Dec. 1997.

[14] J.Voas, "Certifying off-the-shelf software components," *IEEE Comput.*, pp. 53–59, June 1998.

[15] J.Poore et al., "Planning and certifying software system reliability," *IEEE Software*, pp. 88–99, Jan. 1993.

[16] D. Hamlet et al., "Theory of software reliability based on components," in *23rd Int. Conf. Software Engineering*, Toronto, Canada, May 2001.

[17] Petar Popic, Dejan Desovski, Walid Abdelmoez, BojanCukic-Error Propagation in the Reliability Analysis of Component Based Systems-Proceedings of the 16th IEEE International Symposium on Software Reliability Engg. –*IEEE Computer Society*- 2005.

[18] Eric Dubois, Xavier Franch- *International Workshop on models and Processes for the Evaluation of COTS components (MPEC'04)* - ACM SIGSOFT Software Engg. Notes, Sept., 2004, vol.29 no.5 pp. 759-760- IEEE Society Press.

[19] Ralf H. Reussner, Heinz W. Schmidt, Iman H. Poernomo- Reliability Prediction for Component-Based Software Architectures-*The Journal of Systems and software* 66(2003) pp 241-252.

[20] William W. Everett- Software Component Reliability Analysis-*IEEE 1999* pp. 204-211.

[21] Jennifer M. Haddox, Gregory M. Kapfhammer, Christoph C. Michael- An Approach for Understanding and Testing Third Party Software Components-2002, *Proceedings Annual Reliability and Maintainability Symposium*, pp. 293-299.

[22] Vibhu Saujanya Sharma, Kishor S. Trivedi Reliability and Performance of Component Based Software Systems with Restarts, Retries, Reboots and Repairs, 17th *International Symposium on Software Reliability Engineering* (ISSRE'06) IEEE 2006.

[23] B.Littlewood. Software reliability model for modular program structure. *IEEE Transactions on Reliability*, 28(3):241-246, 1979.

[24] D. Mason and D. Woit. Software system reliability from component reliability. In *Proc. of 1998 Workshop on Software Reliability Engineering* (SRE'98), Ottawa, Ontario, July 1998.

[25] L.Krishnamurthy and A. Mathur. The estimation of system reliability using reliabilities of its components and their interfaces. *Proceedings 8th Intl. Symposium on Software Reliability Engineering*, Albuquerque, NM, USA, Nov. 1997.

[26] John D. McGregor, Judith A. Stafford, Il-Hyung Cho, *Measuring Component Reliability*

[27] Jean Dolbec and Terry Shepard, A Component Based Software Reliability Model August 15, 1995.

[28] 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.

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

[30] Saileshwar Krishnamurthy Aditya P. Mathur, On the Estimation of Reliability of a Software System Using Reliabilities of its Components- *IEEE 1997*, pp 146-155.

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

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

[33] 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.

[34] 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.

Biographical 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. His area of interest includes software reliability, neural network, natural language processing, etc.

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.