STRUCTURE BASED RELIABILITY ASSESSMENT OF SOFTWARE USED IN REAL TIME APPLICATIONS

Dissertation submitted to Jawaharlal Nehru University in partial fulfillment of the requirements for the award of the degree of

MASTER OF TECHNOLOGY

in

COMPUTER SCIENCE AND TECHNOLOGY

by

ANJUSHI VERMA

Under the Supervision of

Dr.Tirthankar Gayen

SCHOOL OF COMPUTER & SYSTEMS SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI –110067, INDIA
JULY 2015

जवाहरलाल नेहरु विश्वविद्यालय

JAWAHARLAL NEHRU UNIVERSITY
SCHOOL OF COMPUTER & SYSTEMS SCIENCES
NEW DELHI – 110067, INDIA

CERTIFICATE

This is to certify that the dissertation entitled "**Structure based Reliability Assessment of Software used in Real Time Applications**", submitted by **Anjushi Verma** to the **School of Computer and Systems Sciences, Jawaharlal Nehru University, New Delhi**, is an authentic record of work carried out by her under the supervision of **Dr. Tirthankar Gayen** in partial fulfillment of the requirement for the award of the Degree of **Master of Technology** in **Computer Science and Technology**.

[illegible]

Prof. C. P. Katti

(Dean)

Dean
School of Computer & Systems Sciences
Jawaharlal Nehru University
New Delhi-110067

[illegible] 20/7/15

Dr. Tirthankar Gayen

(Supervisor)

DECLARATION

I hereby declare that the dissertation work entitled **“Structure based Reliability Assessment of Software used in Real Time Applications”** submitted to the School of Computer & Systems Sciences in partial fulfillment of the requirements for the award of degree of **“Master of Technology”** in **“Computer Science & Technology”** in Jawaharlal Nehru University, New Delhi-110067, India is the authentic record of my own work carried out under the supervision of **Dr. Tirthankar Gayen**. This dissertation comprises only my own work. This dissertation is less than 14,000 words in length, exclusive tables, figures and references. The matter personified in the dissertation has not been submitted for the award of any other degree or diploma.

Date: Anjushi Verma

Dedicated to Almighty
God and
My Parents

Acknowledgement

First and foremost, I express my sincere thanks from the core of my heart to the Almighty who has helped and provided me necessary enthusiasm and determination to complete my dissertation. It is with his grace and benevolence that I have been bestowed with a devoted and sincere supervisor and extremely supportive family.

I deeply express my gratitude to Dr. Tirthankar Gayen who has been a source of constant inspiration, support and guidance during all stages of my research work by providing me with valuable suggestions and feedback during the entire tenure. Working under his guidance has been a great learning experience and his deep devotion and patience is extremely exemplary. I would like to follow his footsteps in my further career and life.

I gratefully acknowledge my thanks to Dean of School of Computer & Systems Sciences for providing the necessary infrastructure to pursue my research work and all the teachers for their support and blessings.

Most importantly, none of this world would have been possible without the constant love, patience and support of my family. My heartfelt gratitude for my parents Gaya Prasad Verma and Nirmala Verma, my sister Priyanka Verma and brothers Pramod Kumar and Shashank Verma who encouraged me to pursue my passion and be creative in my field of interest and have always given me the necessary motivation, guidance and feedback without which this work would have not been possible.

I acknowledge each one of those who directly or indirectly, have helped me during the whole period, thus making it a well-rounded experience of learning.

Ms. Anjushi Verma

Abstract

Real time systems are those systems which must guarantee to response correctly within strict time constraint or within deadline like satellite systems, traffic control systems, etc. The severity of consequences resulting from the failure of these systems is usually is very high. Hence, the reliability should be estimated accurately in order to minimize or avoid the losses resulting from the failure of such systems. Failure of real time systems which are used in real life applications can arise from both functional error as well as timing bugs. Therefore, it is necessary to provide temporal correctness of embedded program used in real time applications in addition to providing functional correctness. It has been observed that many approaches do not consider dependency among components, timing bugs, failure dependency and various execution scenarios possible for a given operational profile in the presence of indefinite loops and other instance characteristics. Also, for real time systems, many times it becomes a necessity for a given service to be delivered within the specified time deadline, but many approaches do not explicitly take this issue into consideration. Hence, a versatile and cost effective approach has been developed which closely captures the behavior of the real time software application by considering various aspects like the components dependencies, their failure dependencies (based on the architecture of the system) for various execution scenarios with respect to a given operational profile in the presence of timing bugs. It is expected to serve as an immensely useful approach to the developers, integrators and quality assurers for assessing the reliability of their real time applications.

TABLE OF CONTENTS

Certificate……………………………………………………………………………ii

Declaration…………………………………………………………………………..iii

Dedication…………………………………………………………………..…….iv

Acknowledgement………………………………………………………………...v

Abstract…………………………………………………………………………...vi

Table of Contents………………………………………………………………...vii

List of Figures………………………………………………………………......ix

List of Tables……………………………………………………………………...x

List of Acronyms…………………………………………………………………xi

1. INTRODUCTION……………………………………………………....1

1.1. Software reliability assessment……………………………………………….4

1.2. Motivation……………………………………………………………………….5

1.3. Organization of the Thesis……………………………………………………6

2. OVERVIEW OF SOME SOFTWARE RELIABILITY MODELS………….7

2.1. Black Box Reliability Model…………………………………………………7

2.1.1. Time between Failures Model……………………………………...7

2.1.2. Failure Count Model………………………………………………..9

2.1.3. Fault Seeding Model………………………………………………10

2.2 White Box Reliability Model…………………………………………….…..11

2.2.1. State Based Model…………………………………………….....11

2.2.2. Additive Model……………………………………………………14

2.2.3. Path Based Model…………………………………………………15

3. **RELATED WORK…………………………………………………………..19**

3.1. Literature Survey……………………………………………………………19

3.2. Broad Observation…………………………………………………………..22

3.3. Objective ……………………………………………………………………23

4. **STRUCTURE BASED SOFTWARE RELIABILITY ASSESSMENT FOR REAL TIME APPLICATION………………………………………………24**

4.1. Component Dependency Graph……………………………………………..24

4.2. Proposed Work………………………………………………………………26

4.3. Illustration…………………………………………………………………...29

4.3.1. Calculation of reliability value for *polynomial_series*…………….33

4.3.2. Calculation of reliability value for *expression*……………………..41

5. **RESULTS AND DISCUSSION………………………………………………..45**

5.1. Reliability values for various deadlines……………………………….…45

5.2 Comparison with Yacoub et al. model………………………………………46

6. **CONCLUSION………………………………………………………………49**

7. **REFERENCES……………………………………………………………….51**

8. **APPENDIX……………………………………………………………...……53**

List of Figures

1.1. The Bathtub Curve…………………………………………………………………3

1.2. Revised curve for software……………………………………………………...4

4.1. Component Dependency Graph……………………………………………….25

4.2. Level-0 CDG…………………………………………………………...………30

4.3. Level-1 CDG…………………………………………………………………...31

4.4. Level-2 CDG for *polynomial_series*…………………...…………………………32

4.5. Component Diagram for application………………………………………….33

4.6. CFG for *power*…………………...………………………………………………35

4.7. CFG for *factorial*………………………………………………………………36

4.8. CFG for *polynomial_series*……………………………………………………….38

4.9. CDG for the execution scenario of *polynomial_series*…………………………..…39

4.10. CFG for *expression*……………………………………………………………...41

5.1. The obtained plot between deadline and reliability value for the considered application…………………………………………………………..………46

A.1. CFG for the test code of *polynomial_series*……………………………...………53

A.2. CFG for the test code of *main*……………………………………………..…….56

List of Tables

4.1. Reliability values of components used in *polynomial_series*…………...…………..38

4.2. Reliability of various execution scenario based on number of terms……………….40

4.3. Average reliability values of various components used in *main*………..…………..43

5.1. The deadline and reliability value for the considered application…………………..45

5.2. Comparison with Yacoub et al. model……………………………………………47

A.1. Glue code reliability values for different values of terms…………………………..55

List of Acronyms

CDG	Component Dependency Graph
CFG	Control Flow Graph
COTS	Commercial-Off the Shelf
DTMC	Discrete Time Markov Chain
NHPP	Non-Homogeneous Poisson Process
SMP	Symmetric Multiprocessing

1. INTRODUCTION

In accordance with ANSI, the definition for software reliability is "*The probability of failure free software operation for a specified period of time in a specified environment*" [1]. It can also be stated as a probability of execution of software system without failure within a specified time. Basic terminologies which are being used here are as follows:

- **Error:** A mistake (or a programmer mistake) that may lead to an incorrect result.
- **Fault:** A term that means something imperfect or does not work as expected.
- **Failure:** When system is not able to perform its required function within specified requirements.

Errors can cause fault but not all errors can lead to fault. When the faulty code is executed it results into failure. It is not necessary that all the faults present may lead to failure. There may be some faults which can lead to failure. The application of computer software can be found in various fields. Application of software in safety critical system makes the reliability as a more important attribute of software quality. In safety critical system like real time system, there is a need of high reliability. Real time systems are those systems which must guarantee to response correctly within strict time constraint or within deadline like satellite systems, traffic control system etc. Real time system has also been defined as [10]:

"*A real time computer system is a computer system where the correctness of the system behavior depends not only on the logical results of the computations, but also on the physical time when these results are produced*."

Another definition of real time system is as [2]:

"*Any information processing activity or system which has to respond to externally generated input stimuli within a finite and specified delay*".

In these system generally, if any failure occurs, the severity of consequences resulting from the failure is found to be very high. Thus, these systems require very high reliability and it should be measured accurately. One of real life example of software failure was the explosion of Ariane 5 [3], European rocket. Although, the much of software that was used in *Ariane 5* was also used in *Ariane 4*. In *Ariane 4*, there was no failure but failure occurred in *Ariane 5.* After around 37 seconds of its launching, it veered off its path and lost its control. It was because of software failure. The cause of software failure was the conversion of 64 bit floating point to 16 bit signed integer. Because of overflow error, software failed and there was a loss of around $500 million. This disintegration of *Ariane 5* is perhaps commonly referred by people as one of the most expensive software bugs in the history. Thus, the reliability value of these systems should be estimated accurately in order to minimize or avoid these losses. Failure of embedded systems which are used in real life applications can occur from both errors in function as well as timing bugs [17]. Therefore, it is necessary to provide temporal correctness of the program used in real time applications in addition to provide functional correctness.

Software reliability models can be basically divided into two types [22]:

- Probabilistic
- Deterministic

The fault removals and failure occurrences are considered as probabilistic events in probabilistic type. The various execution paths for the control flow of a program, the operands and operators used are all taken into consideration in deterministic models. The models can be further divided into the following groups [22]:

- Failure rate
- Markov
- Non-homogeneous Poisson process
- Execution path
- Error seeding
- Input domain

- Reliability growth
- Program structure
- Curve fitting
- Bayesian and unified

In hardware, failures can occur because of deterioration in material, environmental factors like aging, rusting etc. As time passes, hardware wears out and it gives an increasing failure rate, as shown in Fig. 1.1.

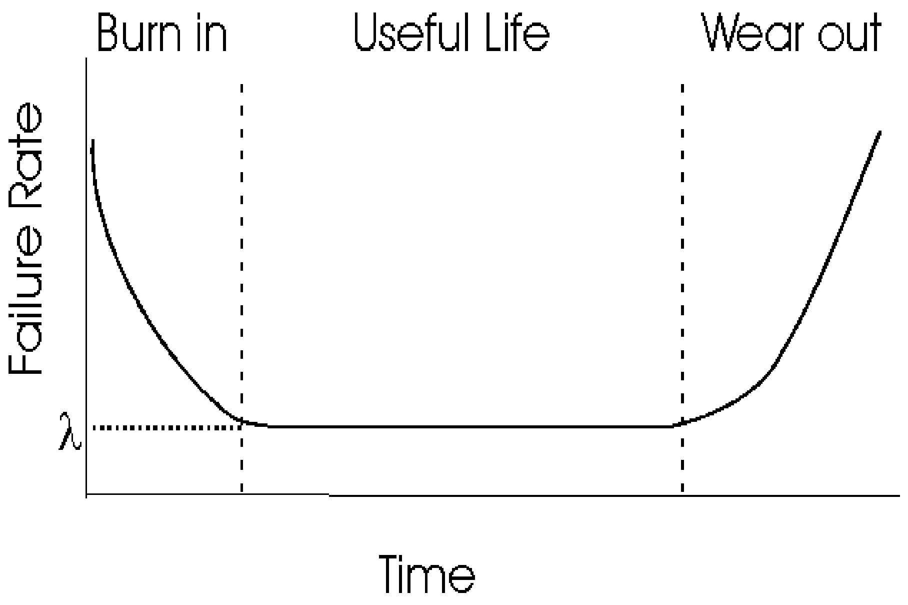

Fig. 1.1: The Bathtub Curve [22]

In Fig.1.1, there are three phases: Burn in phase followed by Useful life phase and Wear out phase. Initially, the failure rate is high in Burn in phase because of some initial defects in the system but when these defects are removed, failure rate decreases and thus reliability increases. In the Useful life phase, failure rate is constant. In the Wear out phase, because of some environmental factors like aging, rusting etc. there may be decay and deterioration of hardware resulting in the increase in failure rate.

In software, failure can be caused by design faults. When any incorrect input or incorrect logic is used, software can cause failure. Fig.1.2 shows the revised curve for software in accordance with some researchers.

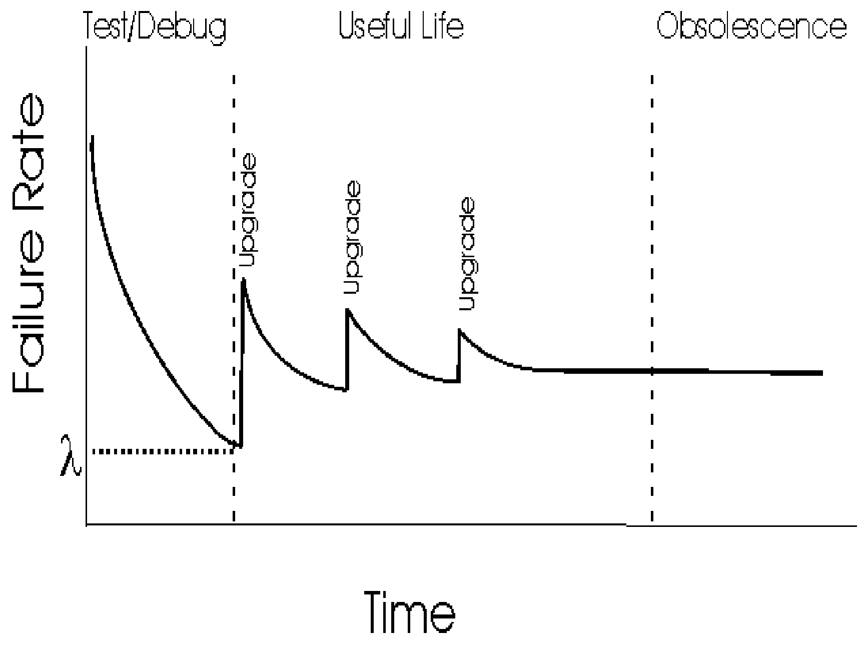

Fig. 1.2:Revised curve for software

In this revised curve for software, there are three phases which are Burn-in, Useful Life and Obsolescence phase. In Burn-in phase, initially failure rate is high but when testing and debugging is done, defects are removed and failure rate decreases. In the useful life, when upgrades are done in the software then there are chances of introducing new faults which leads to hike in failure rate but in the no upgradation phase, the failure rate is constant because no faults are introduced or removed in this phase. In the Obsolescence phase, the failure rate remains constant.

1.1. Software reliability assessment

Software reliability assessment is an action of estimating the reliability of software. In accordance with Yacoub [21], it can be estimated by two ways:

System level reliability

In it, the whole software system is considered as a single unit and then reliability is calculated. Here unit testing is used. But, it may not be very suitable for component based system as it ignores the compositional properties of components.

Component level reliability

In it, the reliability of individual component is calculated and then these individual components' reliability is used to calculate the whole system reliability. Here, integration testing is used and may be more suitable for component based software systems.

Software reliability assessment is very important in real time system like air traffic control systems, satellite systems, etc. In these systems even a small failure can lead to a huge loss. Hence, these systems need to be highly reliable. Since, for real time systems, besides other functional errors, the failure may occur due to timing bugs. Timing bugs can be introduced because of task dependencies which can be caused by task priority, task preferences, resource sharing, etc.[17]. Because of task dependencies, it may be possible that delay in one task can cause delay in other dependent tasks. Therefore, it becomes necessary to provide temporal correctness along with providing functional correctness.

1.2. Motivation

In current time, there is a great need of estimating the reliability of real time safety critical systems because any failure in these systems can lead to a huge loss. These systems require high reliability, since, most of the systems are component based and their attributes like reliability, performance, etc. depends not only on characteristics of individual components but also with their behavior of interacting with each other. Various models like failure count model, time between failure models etc. only focus on observing the way in which the system behaves irrespective of the structure of the system and therefore, these models have lot of uncertainties in measuring the reliability of the system [16]. They treat the software as a black box and do not consider its structure and architecture of the system i.e. they do not consider about the components (composing the system), and their interconnections. These models are not suitable to find out the set of all execution paths for estimating reliability because they do not consider the internal details of the system. Thus, these black box models are not suitable for estimating the reliability of large or complex component based system. So there is a requirement for those models or techniques which are based on architecture and which also consider the behavior of the system. Hence, the structure based approach is suitable, since it relates the system or application reliability to the reliabilities of individual components and system structure.

1.3. Organization of the Thesis

An introduction to software reliability and the importance of its assessment for real time systems has been taken into consideration in section 1. Section 2 presents the overview of some software reliability assessment models. Section 3 presents a survey of related work concerned with the structure based approaches considering various execution scenarios and input domains (or operation profile) for software reliability assessment followed by broad observation, identification of problems and finalization of the objective. Section 4 specifies the proposed approach for the software reliability assessment followed by illustration. Section 5 discusses the results obtained after applying the approach to a suitable application and makes a comparison with the results obtained from Yacoub et al. [21] approach. Finally, section 6 concludes with the discussion of the outcome of research, its advantages and limitations along with the guidelines for future enhancements.

2. OVERVIEW OF SOME SOFTWARE RELIABILITY ASSESSMENT MODELS

In accordance with Pai [9], various models were proposed to measure reliability of software. It may be broadly classified as Black Box Reliability Model and White Box Reliability Model.

2.1. Black Box Reliability Model

In this class of models, the architecture of software is not considered. It has no knowledge of internal working of a system[19]. It requires only failure data. Based on the failure data, it calculates failure rate and then estimates reliability. These models are not suitable for large component based systems. The models which are based on Black Box Reliability model are as follows:

2.1.1. Time between Failures Model

It focuses on time between failures. In it, the time between (i-1)th and ith failure follows a distribution. Its parameters depend on number of faults remaining in a system during this interval. The parameters used may be obtained based on values of time between failures. The various models which are based on Time between Failures Model are as follows:

Jelinski and Moranda (De-Eutrophication Model)

This model is considered to be the foremost model for estimating software reliability [16]. Assuming that the number of faults (which are all independent and equally likely) present in the program causing failure is **N**. The hazard function Z**(t)** during (i-1)th and ith failure is given as:

$$\boldsymbol{Z(t_i) = \phi[N - (i - 1)]}$$

where

$\boldsymbol{\phi}$ is a proportionality constant.

Modification of this model was done by Moranda where faults are removed only when the fatal fault occur. For this model, the hazard function during the ith testing interval is:

$$\boldsymbol{Z(t_i) = Dk^{i-1}}$$

where

$\boldsymbol{D}$ is the fault detection rate,

$\boldsymbol{k}$ is a constant.

Schick and Wolverton Model

In this model assumptions are same as in *Jelinski* and *Moranda* model except that the hazard function is proportional to the current fault content as well as the time elapsed since the last failure. So hazard function **Z(t$_i$)** is [16]:

$$\boldsymbol{Z(t_i) = \{\phi[N - (i - 1)]\}t_i}$$

where

N is the initial faults in the system,

$\boldsymbol{t_i}$ is the cumulative time since the beginning of testing.

Goel and Okumoto Imperfect Debugging Model

All these above models assume that faults are removed with certainty when they got detected but this may not so in reality. Hence, Goel and Okumoto [16] proposed a model where the number of faults in a system is treated as Markov process. So the hazard function **Z(t$_i$)** during (i-1)th and ith failure is given as:

$$\boldsymbol{Z(t_i) = \phi[N - p(i - 1)]\lambda}$$

where

$\boldsymbol{N}$ is the number of initial faults,

$\boldsymbol{p}$ corresponds to the probability of imperfect debugging,

$\boldsymbol{\lambda}$ corresponds to failure rate.

2.1.2. Failure Count Model

This model is based on counting the number of failures during testing intervals. Various models have been proposed on this phenomenon which is as follows:

Goel-Okumoto Nonhomogeneous Poisson Process Model

Assuming N(t) as the cumulative number of failures which is observed by time **t**. Here N(t) is modeled as Non-Homogeneous Poisson process. Based on it, following formula is derived [14]:

$$\boldsymbol{P\{N(t) = y\} = \frac{(m(t))^y e^{-m(t)}}{y!}}, \mathbf{y=0,1,2,\ldots.}$$

where

$$\boldsymbol{m(t) = a\left(1 - e^{-bt}\right), \qquad \lambda(t) = m'(t) = abe^{-m(t)}}$$

where

m(t) corresponds to the expected number of failures observed by the time **t,**

λ(t) corresponds to the failure rate.

Musa Execution Time Model

In this model the assumptions are same as of *Jelinski and Moranda* model but the difference is that the process which is modeled here is the number of failures in a given testing intervals. Hazard function is given as [16]:

$$\boldsymbol{Z(t) = \phi\, f(N - n_c)}$$

where

t corresponds to the execution time used during execution of program,

N corresponds to the number of faults,

n_c corresponds to the number of faults corrected during interval **(0,t)**,

Φ is a constant,

f corresponds to the linear execution frequency.

Shooman Exponential Model

It is similar to *Jelinski* and *Moranda* model. Hazard function is given as[16]:

$$\boldsymbol{Z(t) = k[\frac{N}{I} - n_c(\tau)]}$$

where

$\boldsymbol{\tau}$ corresponds to the time to debug from the start of integrating system,

$\boldsymbol{t}$ corresponds to the operating time,

$\boldsymbol{I}$ corresponds to the total number of instruction,

$\boldsymbol{k}$ is a constant.

Generalised Poisson Model

This model [16] is a variation of the NHPP model of Goel and Okumoto. The mean value function it assumes is as:

$$\boldsymbol{m(t_i) = \phi(N - M_{i-1})t_i^{\alpha}}$$

where

$\boldsymbol{M_i}$ corresponds to the total number of faults removed upto the (i-1)st debugging interval,

$\boldsymbol{\Phi}$ is a constant,

$\boldsymbol{\alpha}$ is a constant used to rescale time $\mathbf{t_i}$.

2.1.3. Fault Seeding Model

In this model, a number of faults are seeded in the program. When testing is completed, the number of indigenous faults and exposed seeded faults are counted. By using the maximum likelihood estimation, software reliability is to be estimated. The model which is based on Fault Seeding Model is as follows:

Mills Seeding Model

In this model [16], a number of known faults are seeded in the program to be tested and then this program is tested for some amount of time. The original indigenous faults can be estimated from number of indigenous and seeded faults uncovered during the testing.

2.2. White Box Reliability Model

In this model, whole architecture is known and it has access to source code. These are suitable for estimating the reliability of component based system. Three different approaches are recognized to combine the architecture of the software with the failure behavior [12]:

- **State based model:** This model uses the control flow graph to represent software architecture and transitions follow Markov process.
- **Additive model:** It does not consider the architecture of the application explicitly so it is not commonly used.
- **Path based model:** It computes reliability of software considering various possible execution paths existing in the program. The execution paths may be determined by executing the application, or algorithmically.

2.2.1. State Based Model

The models which are based on State Based Model are as follows:

Gokhale et al.'s State Based Model

It describes the architecture using absorbing discrete time Markov Chain. This model [18] either assumes that reliabilities of components are known or estimates it using Software Reliability Growth Model. Here the control flows between components are assumed to be a Markov process. Time dependent failure intensity $\boldsymbol{\lambda_i(t)}$ and cumulative expected time spent in component per execution of application is given. So the reliability of each component **i** is calculated as:

$$R_i = \prod_{i=1}^{n} e^{-\int_0^{V_i T_i} \lambda_i(t)}$$

By using the components reliability, overall reliability is to be calculated as:

$$R_{sys} = \prod_{i=1}^{n} R_i$$

Cheung's State Based Model

This model [15] assumes that there is a single entry and exit node at which the execution begins and terminates respectively. Cheung model is used which measures software reliability considering components utilization and their reliabilities. The transitions among the components is represented by an absorbing DTMC using transition probability matrix **P** = **[p_{ij}]** where **p_{ij}** is transition probability from component **i** to component **j**. In a control flow graph, two states **F** and **C** are added as the terminal states. **C** state represents state of correct output and **F** state represents failure. For each node **N_i**, a directed edge (**N_i, F**) is created having transition probability (**$1-R_i$**). It represents the occurrence of an error in execution of node **N_i.** Here errors do not compensate each other so a failure in node **N** will lead to an incorrect system output without considering the sequence of nodes executed afterwards. It can be shown by the transition to terminal state **F**. The original transition probability between node **N_i** and **N** is represented by **$R_i p_{ij}$**, which shows node **i** is executed properly and the transition occurs from node **i** to **j**. For exit node N, there is a directed edge (**N_n, C**) which is used to show correct termination at the exit node with transition probability **R_i**. Failure is represented as creating a directed edge with a transition probability (**$1-R_i$**). Thus, from the initial state of Markov process the probability of reaching the terminal state **C** is the reliability of a program. Let **R_i** is the reliability of node **N_i** and **p_{ij}** is the transition probability from node **N_i** to **N_j**. **P_{ij}** = 0 if there is no edge between (**N_i, N_j**). The states of transition matrix is {**C,F,N_1,N_2,N_3…..N_n**} so the transition matrix **$P^\wedge$** is represented as:

$$P^{\wedge}= \begin{array}{c|cccccccc} & \mathbf{C} & \mathbf{F} & \mathbf{N_1} & \mathbf{N_2} & \mathbf{\dots} & \mathbf{N_j} & \mathbf{\dots.} & \mathbf{N_n} \\ \hline \mathbf{C} & 1 & 0 & 0 & 0 & \dots & 0 & \dots & 0 \\ \mathbf{F} & 0 & 1 & 0 & 0 & \dots & 0 & \dots & 0 \\ \mathbf{N_1} & 0 & 1-R_1 & 0 & R_1P_{12} & \dots & R_1P_{1j} & \dots & R_1P_{1n} \\ \mathbf{.} & . & . & . & \dots & \dots & \dots & \dots & \dots \\ \mathbf{N_i} & 0 & 1-Ri & 0 & R_iP_{i2} & \dots & R_iP_{ij} & \dots & R_iP_{in.} \\ \mathbf{.} & . & . & . & \dots & \dots & \dots & \dots & \dots \\ \mathbf{N_{n-1}} & . & 1-R_{n-1} & 0 & R_{n-1}P_{(n-1)2} & & R_{n-1}P_{(n-1)j..} & & R_{n1}P_{(n-1)n} \\ \mathbf{N_n} & R_n & 1-R_n & 0 & 0 & \dots & 0 & \dots & 0 \end{array}$$

Q is the matrix obtained from **P**^ after deleting all rows and columns corresponding to **C** and **F** states.

$$Q= \begin{array}{c|cccccc} & \mathbf{N_1} & \mathbf{N_2} & \mathbf{\dots} & \mathbf{N_j} & \mathbf{\dots.} & \mathbf{N_n} \\ \hline \mathbf{N_1} & 0 & R_1P_{12} & \dots & R_1P_{1j} & \dots & R_1P_{1n} \\ \mathbf{.} & \dots & \dots & \dots & \dots & \dots & \dots \\ \mathbf{.} & \dots & \dots & \dots & \dots & \dots & \dots \\ \mathbf{.} & \dots & \dots & \dots & \dots & \dots & \dots \\ \mathbf{N_i} & 0 & R_iP_{i2} & \dots & R_iP_{ij} & \dots & R_iP_{in.} \\ \mathbf{.} & \dots & \dots & \dots & \dots & \dots & \dots \\ \mathbf{N_{n-1}} & 1-R_{n-1} & R_{n-1}P_{(n-1)2} & \dots & R_{n-1}P_{(n-1)j} & \dots & R_{n-1}P_{(n-1)n} \\ \mathbf{N_n} & 0 & 0 & \dots & 0 & \dots & 0 \end{array}$$

So the reliability is calculated as:

$$R = P^{\wedge n}\ (N_1, C)$$

S is a matrix such that:

$$S=1+Q_1+Q_2+Q_3\dots+Q_n$$

So, $R=S(1,n)\ R_n$

The reliability is calculated by adding the reliability value of all paths existing in the control flow graph.

Littlewoods State Based Model

It is one of the earliest architecture based software reliability model [6]. Here there is an assumption that irreducible SMP can describe the software architecture of continuously running application. It consists of a finite number of states and transfer of control between states is described by probability $\mathbf{P_{ij}}$=**Pr**{transfer of control from node **i** to node **j**}. When module **i** is executed, failure is caused according to a Poisson process with parameter $\boldsymbol{\lambda_{i.}}$,$\mathbf{V_{ij}}$ is probability of failure occurring when node **i** calls node **j**. Here the failure rate of Poisson process is given by:

$$\boldsymbol{\lambda_s = \sum\nolimits_i a_i \lambda_i + \sum\nolimits_{ij} b_{ij} V_{ij}}$$

where,

$$\boldsymbol{a_i = \frac{\prod_i \sum_j p_{ij} m_{ij}}{\sum_i \prod_i p_{ij} m_{ij}}}$$

corresponds to the proportion of time spent in module **i** and

$$\boldsymbol{b_{ij} = \frac{\prod_i p_{ij}}{\sum_i \prod_i \sum_j p_{ij} m_{ij}}}$$

is the frequency of transfer of control between **i** and **j**.

2.2.2. Additive Model

The model which is based on Additive Model is as follows:

Everett Model

It is also used for component based software system [26]. It estimates the reliability by using the individual component's reliability. Here extended execution time model is used to analyze the reliability of each component whose parameters can be estimated from the information on the way in which test cases stress each component and also from the properties of software. There is a superimposition of components reliabilities for overall system in the combined model.

2.2.3. Path Based Model

The models which are based on Path Based Model are as follows:

Shooman Model

It is one of the earliest models of path based approach. It estimates reliability by using the frequencies of path with which different paths are run. Here the total number of failure in N test runs is given by [12]:

$$\boldsymbol{n_f = \sum_{i=1}^{m} N\, f_i q_i}$$

where

Nf$_i$ is the total number of frequency of execution of path i.

Thus the system probability of failure on any test run is given by

$$\boldsymbol{q_0 = \lim_{n\to\infty} n_f/N = \sum_{i=1}^{m} N\, f_i q_i}$$

Input Domain Based Model

In this model, various test cases are generated from input distribution. Since there is a difficulty while measuring the input distribution, so the input domain is partitioned into the set of equivalence classes by the various models in this group. An equivalence class is generally associated with a program path. Models which are based on this approach are as follows:

Nelson Model

Here reliability of software is measured by executing the program for n inputs. These inputs are randomly chosen from the input domain sets. If $\mathbf{n_e}$ is the number of inputs that resulted in execution failures, **n** is the number of inputs then reliability measured as:

$$\boldsymbol{R = 1 - \left(\frac{n_e}{n}\right)}$$

Here the reliability function remains constant but the estimate of its can be changed.

Gayen and Misra model

This model [23] is an improvement of Weiss model[20]. In Weiss model, there were no guidelines for choosing test cases and finding the occurrence of operational error. But in Gayen and Misra model, this problem is solved. Here input domain is divided into two sub domains: Operational error sub domain and logical error sub domain. Weiss' model extends Nelson's definition of "probability of successful execution **R(P)**" of program with respect to a given specification(S) is given as:

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

Here **p(n)** is the probability that input n is submitted to **p**.

$$\begin{aligned} \boldsymbol{a(n)} &= \mathbf{0}\ if\ P(n) = S(n) \\ &= \mathbf{1}, otherwise \end{aligned}$$

T is a set of test cases and $\mathbf{p_T(t)}$ is a degree of representativeness to **T**. Suppose **T** is obtained by selecting inputs randomly according to distribution **p**, then **T** shows operational profile so, $\mathbf{p_T(t)=1/|T|}$**.** Therefore generalized logical reliability is:

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

where

$\mathbf{d_\alpha}$**(Sp ,P, t**) shows α discrepancy between **S(p)** and **P** at **t**.

In this model, the overall reliability is calculated as the product of operational reliability and logical reliability.

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

Operational reliability is calculated by finding the probability of non-occurrence of operational error.

Ramamurthy and Bastani Model

It gives a measurement of the conditional probability for a program to be correct for all possible inputs assuming that it to be correct for a given set of inputs. In accordance with this model [3],

P {the program is correct for all points in [a, a + V | it is correct for the test cases having successive distances $\mathbf{x_j}$, where j = 1, …, n - 1}

$$= e^{-\lambda v} \prod_{j=1}^{n-1} \left[\frac{2}{1 + e^{-\lambda x_j}}\right]$$

where

λ is a parameter deduced from the complexity of the source code.

Krishnamurthy and Mathur's Path Based Model

In this model, it is assumed that reliabilities of components are known in advance. The reliability is calculated by averaging the path reliabilities over all the test cases run on software whose reliability is to be estimated. Path reliability is calculated from the sequence of components involved in a particular path when the set of test cases are executed. If each path has reliability $\mathbf{R_p}$ and **T** is set of test cases then overall system reliability $\mathbf{R_{sys}}$ is defined as [12]:

$$R_{sys} = \frac{\sum_{p \in path(p)} R_p}{|T|}$$

Thus by finding the set of execution paths in a software system, path reliability is to be calculated which is used in estimating the overall system reliability.

Yacoub’s Scenario Based Reliability Assessment Model

Yacoub [21] introduced a reliability model named as Scenario Based Reliability Analysis. It is a path based approach which is based on execution scenarios. An algorithm is proposed to analyze the reliability of a component based system using components, interface and link reliabilities. Using scenarios, a component dependency graph is constructed. Based on it, an algorithm is given. This algorithm describes that breadth expansion of a path gives logical OR i.e. reliability is calculated as a sum of all components reliability on the same level along with a transition probability. The depth of each path represents the components arranged in series which represents the logical AND and it is the multiplication of reliabilities. The main problem is that it does not consider the failure dependency between components.

Since, most of the systems are component based and their attributes like reliability, performance, etc. not only depends on characteristics of individual components but also on their behavior of interacting with each other. Various models like failure count model, time between failure models etc. only focus on observing how the system behaves instead of considering the structure of the system. As a result, these models may have lot of uncertainties in measuring the reliability of the system [16]. They treat the software as a black box and do not consider its structure and architecture of the system. In other words these approaches do not consider the components composing the system, their interconnections etc. These models may fail to obtain the set of all execution paths for estimating reliability because they do not consider the internal structure of the system. Thus, these black box models may not be suitable for estimating the reliability of large component based system. So there is a need of those models or techniques which are based on architecture and which considers the behavior of the system based on various execution instances. Hence, the structure based approach is suitable since it relates the system or application reliability value to the reliability values of individual components and system structure. The next section outlines the existing work concerned with various approaches based on the structure, execution scenarios considering various input domain/operational profile for estimating the reliability of component based software.

3. RELATED WORK

Since, most of the real time applications are component based and their attributes like reliability, performance, etc. not only depends on characteristics of individual components but also on their behavior of interacting with each other. Various models like failure count model, time between failure models etc. only focus on observing how the system behaves instead of considering the structure of the system. As a result, these models have lots of uncertainties in measuring the reliability of the system [16]. They treat the software as a black box and do not consider its structure and architecture of the system. In other words these approaches do not consider the components composing the system, their interconnections etc. These models may fail to obtain the set of all execution paths for estimating reliability because they do not consider the internal structure of the system. Thus, these black box models may not be suitable for estimating the reliability of large or complex component based system. So there is a requirement of those models or techniques which are based on architecture and which considers the behavior of the system based on various execution instances. Thus, a survey of various approaches based on the structure, execution scenario considering various input domain/operational profile for estimating the reliability of component based software was performed.

3.1. Literature Survey

From the survey, the various approaches which were found based on the structure, execution scenario considering various input domain/operational profile for assessing the reliability of component are outlined as follows:

According to Biswas et al. [17], in embedded programs used in real time applications, various execution dependencies exist because of task priority, task precedence etc. Failure of these programs can occur from both errors in function as well as timing bugs. Therefore it is necessary to provide temporal correctness of embedded program used in real time applications in addition to provide functional correctness.

Gayen [22] analyzed the shortcomings in the conventional failure rate based models and proposed an error based model for predicting the minimum reliability of software. It is given that in the useful life of software, when there are no upgrades, various authors have contradicting opinions. Some researchers say that reliability depends upon time but some other researchers say that it is time independent. According to Gayen, if the software is not modified, the reliability will not change with time. However the reliability can be changed based on the changes in the operating environment. He proposed an error based reliability model for predicting the minimum reliability of software and showed that the reliability value varies with the number of errors debugged.

Yacoub et al. [21] made execution scenario based reliability analysis to propose a path based approach. The algorithm analyzes the reliability of a component based system using components, interface and link reliabilities. Using scenarios, a component dependency graph is constructed. Based on it, an algorithm is given. This algorithm describes that breadth expansion of a path gives logical OR. The depth of each path represents the components which are arranged in series which represents the logical AND. The main problem is that it does not consider the failure dependency between components.

According to Goel [4], various software reliability growth models were proposed like Time Between Failure model, Failure Count model etc.. But these models are not suitable for all type of system because of various assumptions. Some of the models assume that all faults in program contribute equally to reliability of software which is not realistic because occurrence frequencies of different types of faults may be different. These models assume that all failures are independent, it can be corrected in negligible time and debugging is done perfectly which is not realistic.

Singh et al. [5] proposed a model for component based system to measure the reliability through path propagation probability, component usage ratio and component impact factor. Component architecture graph is constructed to find out the set of all execution paths. The study assumes that individual component reliability is known in advance. Here, the component impact factor determines the impact of each component on the reliability of a system. The component whose impact factor is high, the effect of it will be more on reliability. Component usage ratio is calculated to find the ratio of total component execution time over the total software system execution time. It does not give the exact value of usage ratio. It varies with the varying set of input data which was the main drawback of this approach.

According to Gayen et al. [24], a unique method was proposed which is based on the execution scenario analysis for COTS component based software application. In it, maximum and minimum reliability values are computed from the execution scenarios. The reliability bounds although provides us the range of reliabilities values irrespective of any operational profile yet for practioners, it becomes sometimes necessary to use the exact reliability values for a specified operation (i.e.using a specific operation profile concerned with the given operation.) Again sometimes it becomes difficult to obtain the exact operation profile for a given operation.

Gayen et al. [23] has given reliability assessment of elementary COTS software component. Sometimes it becomes very difficult to select test cases to execute all possible executable paths of the program so a new approach was developed. In it, input domain is divided into operational error sub domain and logical error sub domain and errors are identified. From control flow graph, test cases are selected and these are executed to obtain the alpha discrepancy between the program and its specification. Reliability was calculated by multiplying the probability of non-occurrence of operational error and the probability of logical correctness. Here, the timing bugs for real time systems were not taken into consideration.

Cheung [15] introduced a model that measures a reliability of a service which is to be provided to a user community. The reliability of service is dependent on the reliability of the components and user profile. So user oriented software reliability is to be used to measure the reliability with respect to a user environment.

Weiss [20] introduced a tolerance function which measures the acceptable level of correctness. Alpha discrepancy is used to calculate the tolerance function between the given specifications and program. By using these parameters, reliability is to be estimated. Here it becomes very difficult to find out the test cases and operational error because there is no guideline for selecting it.

Goswami et al. [25] proposed a method to measure the reliability of component based software system. In it, it is given that the component whose execution time is more, contribute more on the reliability of a software system than the component having less execution time. It measures the reliability by considering individual component reliability, component usage ratio etc. It does not consider the dependency among components. According to Hsu et al. [8], an adaptive framework is used to use path testing into assessment of reliability of a modular software system. The overall system reliability is calculated by doing summation of all the individual path reliability. Here also a sensitivity analysis is used to find out those components which are more important than others. In it, dependency among faults was not considered.

Xiaoguang et al. [14] proposed a general method for estimating reliability of component based software system. In it a general model-component probability transition diagram is used which enables the reliability tracing through the component based software process.

Nautiyal et al. [13] proposed an approach for measuring the reliability of component based software architectures. The proposed approach considers the reliability of components and paths to calculate overall system reliability, depending on usage time of components and executions paths.

3.2. Broad Observation

From the survey, it was found that there are various assumptions which are made in black box models like failures are independent, perfect debugging and failure can be corrected in a negligible time [16]. These assumptions are not realistic as most of the software does not follow these assumptions. Since component/system execution time depends on various instance characteristics like input data, execution scenario, loops (indefinite) etc. and hence it varies with various instances of execution of the program [5]. Therefore, it becomes very difficult to obtain the exact usage ratio for component usage ratio.

Dependency among the components has not been considered in many approaches [21]. The reliability bounds although provides us with the range of reliability values irrespective of any operational profile yet for practioners, it becomes sometimes necessary to use the exact reliability values for a specified operation (i.e. using a specific operation profile concerned with the given operation. Again, sometimes it becomes difficult to obtain the exact operation profile for a given operation [13].

Although, several software reliability models were proposed to measure the reliability of a system, again in many of these models timing bugs are not considered which an important issue of real time applications.

3.3. Objective

From the survey, it has been observed that many approaches do not consider dependency among the components, timing bugs and failure dependency for various execution scenarios possible for a given operational profile considering various instance characteristics. Many assumptions are also not realistic in the sense that the actual software behaves differently from what has been assumed or taken into consideration in the existing Black box reliability models. Considering these aspects, the main objective is to develop a structure based reliability assessment approach by considering the components dependencies, their failure dependencies based on the architecture of the system for various execution scenarios with respect to a given operational profile in the presence of timing bugs.

4. STRUCTURE BASED RELIABILITY ASSESSMENT APPROACH

With the increasing reuse of components in large complex software, the focus would be on estimating the reliability of component based software system. Complex software consists of various components and these components can be reused in any software thus it reduces time and cost. Real time application also consists of various components. But Black box models are not suitable for large component based software as they do not consider the components and their interactions [2]. Therefore the White box structure based models were proposed which consider the architecture of the system (For example, the structure of components, their interconnections etc.) Based on the architecture of the system, these models find out the set of execution paths in the software system to estimate reliability. Here for estimating reliability of whole application, reliability of individual components involved in a system, is calculated.

So, reliability of the application is:

$$R_{app} = f(RC_1, RC_2, RC_3, \ldots.. RC_n)$$

where $\mathbf{RC_1}, \mathbf{RC_2}, \mathbf{RC_3}, \ldots.. \mathbf{RC_n}$ are the reliability values of first, second, third, ...nth components respectively.

4.1. Component Dependency Graph

Architecture of a software can be represented using component dependency graph. Component dependency graph is constructed among the components.

According to S. Yacoub [21], it is defined as:

It consists of (N,E,S,T)

where

E corresponds to the set of edges in graph: **E**={**e**},

N corresponds to the set of nodes in graph: **N**={**n**},

S corresponds to the start node,

T corresponds to the terminating node.

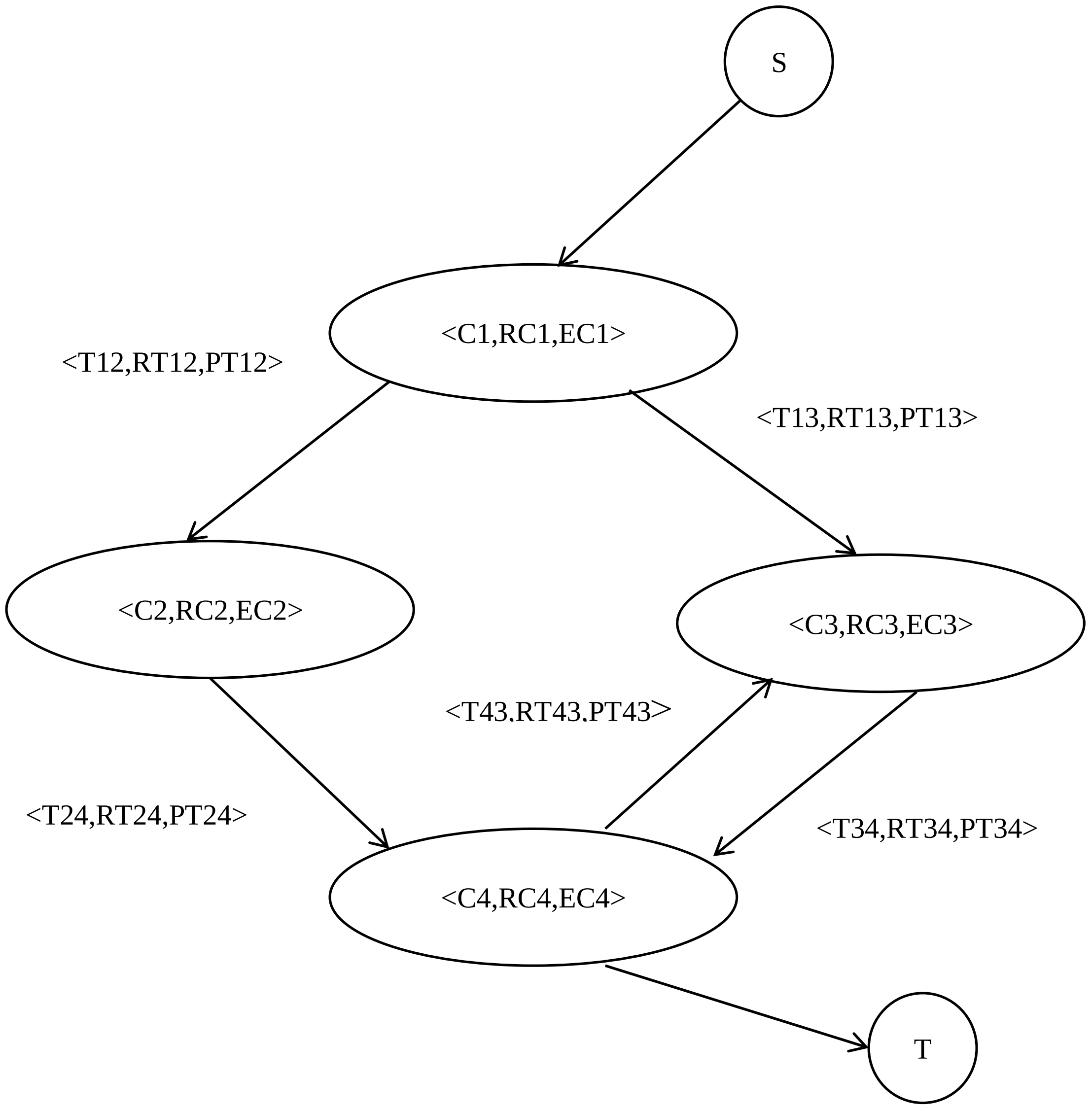

Fig 4.1: Component Dependency Graph[21]

N node is defined as a tuple <***C_i*, *RC_i*, *EC_i***>,

where

RC_i corresponds to the individual component reliability,

C_i corresponds to the individual component,

EC_i corresponds to the individual component execution time.

A directed edge is defined as tuple <$\boldsymbol{T_{ij}}$, $\boldsymbol{RT_{ij}}$, $\boldsymbol{PT_{ij}}$>

*w*here,

> $\mathbf{RT_{ij}}$ corresponds to the transition reliability from node $\mathbf{n_i}$ to $\mathbf{n_j}$,
>
> $\mathbf{T_{ij}}$ corresponds to the transition name from node $\mathbf{n_i}$ to $\mathbf{n_j}$,
>
> $\mathbf{PT_{ij}}$ corresponds to the transition probability from node $\mathbf{n_i}$ to $\mathbf{n_j}$.

From the component dependency graph, various parameters are calculated. Breadth expansion of the graph gives a logical OR of reliability and depth expansion gives logical AND of reliability.

4.2. Proposed Work

Reliability assessment plays an important role in real time applications as in these applications, severe of consequences resulting from failure is very high. Therefore an approach for estimating the reliability of real time application considering the dependencies among components, failure dependency and various execution scenarios with respect to a given operational profile in the presence of timing and functional bugs is proposed. Here, a function is considered as a component. A function may invoke other sub functions which are known as subcomponents. In this approach, a function or a composite component is decomposed into subcomponents until elementary component (a function which do not invoke any other user defined function) is obtained. Since, here it is concerned with the reliability assessment of the real time application and not the system, hence the operating environment/system is considered to be idle. Hence, the system functions used over here are considered to be fully reliable.

A program may consist of various functions and sub functions. Each function is assumed as a component in CDG. CDG is constructed at different level. Here bottom up approach is used to estimate the reliability of application by calculating the reliability of each execution scenario. The proposed algorithm *find_Rel* provides the average reliability value of the application by using structure based reliability assessment for real-time application.

```
Algorithm: find_Rel(level, component)
{
    Step 1:if component is elementary
    {
    if(level = = 1)
    return elem_Rel(component)*p($T_{ji}$<=deadline);
    else
    return elem_Rel(component);
    }
    Step 2:Decompose the given composite component into subcomponents and construct
            CDG.
        Step 2.1:  Repeat for all component i in CDG,
        {
        Rel[i]=find_Rel(level+1,i)
        }
        Step 2.2:  Repeat for each execution scenario j,
        {
            Step 2.2.1: Repeat for all component i present in the execution scenario j,
            {
            if (level!= 1)
            R[i]=Rel[i];
            else
            R[i]=Rel[i]*p($T_{ji}$<=deadline);
            }
            Step 2.2.2: if component k comes before i in the execution scenario,
            P[i|k]=R[i]*$P_{i,k}$;
            else
          P[i|k] = 1;
```

When ***k****=C (i.e where* **C** corresponds to a conceptual ideal component assumed to be present in the execution scenario before the first component of the execution scenario*)* ***$P_{i,k}$=1;***

Since components dependency is also considered so transition probability among components is to be calculated. It is represented as **$P_{i,k}$** which is the transition probability from component **i** to **k.**

P[i|k]corresponds to the conditional probability of proper execution of the execution scenario upto component **i**, assuming that the execution scenario upto component **k** has executed properly**.**

Step 2.2.3**:***Move to the last node executed in CDG and find out its reliability value in accordance with Markov model recursively by using the relation*

P[i] = P[i | i_{prev}]*P[i_{prev}] ;

P[i_{prev}]=1 when **i** corresponds to the first component used in the execution scenario **j.**

where

> **i_{prev}** corresponds to the previous component in the execution scenario before **i,**
>
> **$P_{i,k}$** corresponds to the transition probability from component **i** to component **k**,
>
> **P[i]** corresponds to the probability of proper execution of the execution scenario upto component **i.**

Step 2.2.4**:** $Rel_{Scenario}$[j]=P[$\mathbf{i_{last}}$]* Rel_{glue};

where

> ***Rel_{glue}*** corresponds to the reliability value of the glue code used in the execution scenario **j,**
>
> **i_{last}** corresponds to the last component of the execution scenario **j.**

}

```
        Step 2.3: Find the average reliability value comp_Rel=
                  $\sum_{\forall sceanrio\ j} p(occ\ of\ sceanrio\ j) * Rel_{sceanrio}[j]$;
    Step 3: return comp_Rel;
}
```

Algorithm *elem_Rel(component)* returns the reliability value of elementary component by using the method proposed in Gayen and Misra model [23] (discussed under section 2.2.3).

4.3. Illustration

A simple application which calculates either the roots of a quadratic equation or the value of a polynomial depending upon the users choice within a deadline of 6 units of time where 1 unit=1 CPU clock cycle. The input domains considered for this application are as follows:

Range for 'terms'=0 to 6
Range for 'x'=1 to 10
Range for 'a'=1 to 23,000
Range for 'b'=1 to 45,000
Range for 'c'= 1 to 23,000

Here, it is considered that the inputs are provided randomly from the input domain and in accordance with Weiss [20], if the test suite T is constructed by choosing inputs randomly in accordance with the distribution p (like using a random number generator), then T represents an operational profile, and theoretically may then be treated as a uniformly distributed set.

Therefore, each input in the input domain has equally likely chances of occurrence.Since, here reliability assessment was done for component based system, so if a component is not able to deliver its service or perform its task within the application's deadline, further components are not considered for estimating reliability, since the application fails. If all the components are performing their task within the specified deadline then there would be less chance of failure and hence the reliability value would be high. Using the proposed approach the reliability value of each component is evaluated and further, these obtained reliability values are used to obtain the reliability value of the entire application.

The application was run 1000 times to note the number of times the tasks are getting completed within deadline for inputs taken randomly from the input domain. The data obtained from it is used to calculate the probability of completion of task within the deadline.

For a program which consists of various functions and sub functions are as:

main, polynomial_series, expression, power ,factorial.

In Level-0 CDG, ***main*** represented as $C_{0.}$

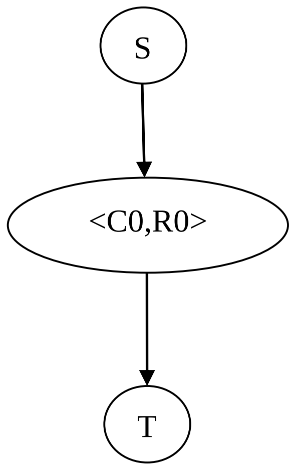

Fig. 4.2:Level-0 CDG

In Level-1 CDG, ***main*** function is explored and the CDG is constructed accordingly. It consists of two functions, one called ***polynomial_series*** is used for evaluating the series for the values of **x** and **n** provided as input.

The formula is as :

$$1+\frac{x}{1!}+\frac{x^2}{2!}+\frac{x^3}{3!}\dots+\frac{x^n}{n!}$$

where

‘x’ and ‘n’ are finite values.

$$\frac{b^2 - 4ac}{2}$$

and the other called ***expression*** is used to return the value of the expression for the values of **a, b** and **c** provided as input. The returned value is used to calculate real roots of the quadratic equations.

$$\begin{tikzcd}
 & S \arrow[dl] \arrow[dr] & \\
\langle C_{00},RC_{00}\rangle \arrow[dr] & & \langle C_{01},RC_{01}\rangle \arrow[dl] \\
 & T &
\end{tikzcd}$$

Fig.4.3:Level-1 CDG

where

C_{00} represents *polynomial_series*,

C_{01} represents e*xpression*,

S is the start node,

T is the terminal node.

Level-2 CDG

In this level, the components of Level-1 CDG are explored. Component *polynomial_series* contains various subcomponents like *power* and *factorial*.

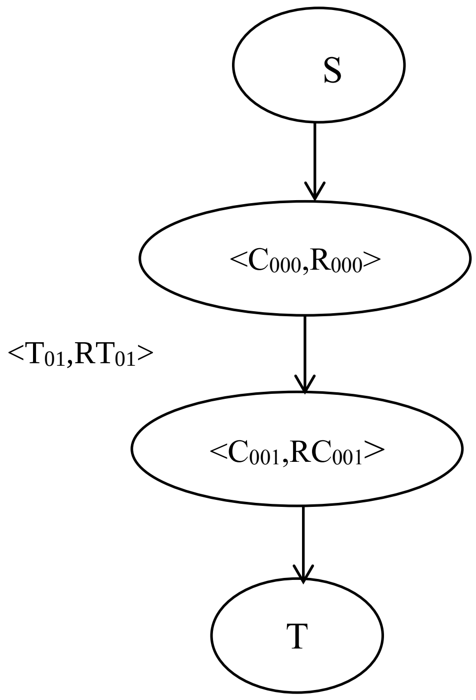

Fig.4.4: Level-2 CDG for *polynomial_series*

where

C_{000} represents *power*,

C_{001} represents *factorial,*

T_{01} represents transition name from C_{000} to C_{001},

RT_{01} represents transition reliability from C_{000} to C_{001}.

Component diagram for the above explained example application is shown in Fig. 4.5.

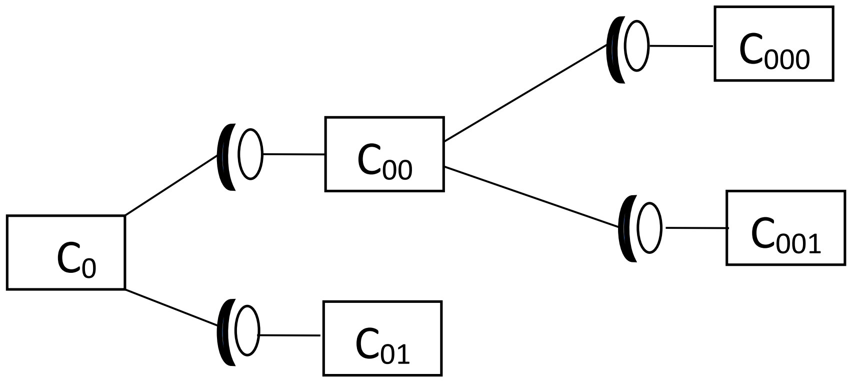

Fig.4.5: Component Diagram for application

In this component diagram,C_0 is requesting service from C_{00} and C_{01}. C_{00} is requesting service from C_{000}, C_{001}.

where

C_0 corresponds to the *main*,

C_{00} corresponds to the *polynomial_series*,

C_{01} corresponds to the e*xpression,*

C_{000} corresponds to the *power,*

C_{001} corresponds to the f*actorial*.

Reliability of elementary components are calculated by using the algorithm *elem_Rel.*

4.3.1. Calculation of reliability value for *polynomial_series*

For the polynomial_series

$$1+\frac{x}{1!}+\frac{x^2}{2!}+\frac{x^3}{3!}\dots+\frac{x^n}{n!}$$

'terms' is taken as **'int'** type and 'x' is taken as **'long int'** so the range for 'int' and 'long int' is:

int: **-32,768** to **32,767**

long int: **2,147,483,648** to **2,147,483,647**

Valid output for ***power*** and ***factorial*** should be within the range of ***integer.***

Based on these ranges, the probability of operational correctness is calculated using the proposed approach.

Calculation of probability of operational correctness for elementary components

For ***power*** component:

Range for ‘terms’=0 to 6

Range for ‘x’=1 to 10

So total number with in this range=70

When ‘term’= 5-6 and ‘x’=8-10, it will give overflow error.

Thus,

Range of integer data causing overflow error for ‘terms’=5-6

Range of integer data causing overflow error for ‘x’=8-10

Total number of times power will give overflow error= 6

Probability of occurrence of data giving overflow error=6/70=0.086

Probability of occurrence of data not giving overflow error =1-0.086=0.914

Probability of operational correctness of *power* is 0.914

For ***factorial*** component:

Range for ‘terms’=0 to 6

Range of integer data causing overflow error for ‘terms’=0

Probability of occurrence of data giving overflow error =0

Probability of occurrence of data not giving overflow error=1-0=0

Probability of operational correctness for *factorial*=1.

Calculation of probability of logical correctness for elementary components

Calculation of probability of logical correctness for ***power*** is as follows:

Source code for *power*

```
double power(int num, int pow)
{
    1.  int i=1;
    2.  double sum=1.0;
    3.  while(i<=pow)
        {
    4.   sum=sum*num;
    5.   i++;
        }
    6.  return sum;
}
```

CFG for *power*

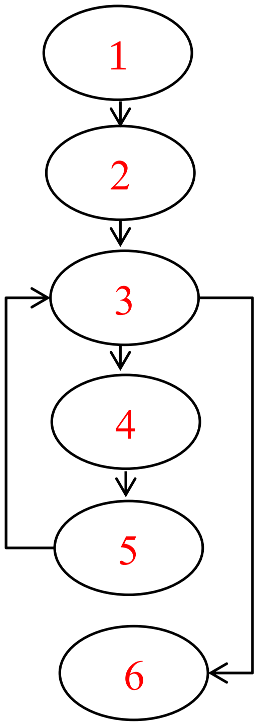

Fig. 4.6: CFG for power

There are 2 linearly independent paths.

1-2-3-6
1-2-3-4-5-3-6

Range of ‘terms’=0 to 6
Range of ‘x’=1 to 10
There are 2 test cases obtained to cover these two linearly independent paths which are as:

When terms= -99 and x=6 (path will be covered as 1-2-3-6):

Expected output=1
Obtained output=1
So difference=| Expected output- Obtained output|=0

When terms =5 and x=7 (path will be covered as 1-2-3-4-5-3-6):

Expected output=2401

Obtained output=2401
So difference=| Expected output- Obtained output|=|2401-2401|=0

Considering the tolerance allowed i.e α= 0.9

R_{log} = 1- 1/2{(0+0)/ 0.9} =1- 0

= 1

It will give power in integer and there will be no precision error in any case while calculating power so logical correctness of *power* is 1.

Calculation of probability of logical correctness for ***factorial*** is as follows**:**

Source code for *factorial*

```
double factorial(int terms)
 {
    1.  int i;
    2.  double f=1;
        for ( a i=1; b i<=terms; c i++)
    3.  f=f*i;
    4.  return(f);
}
```

CFG for *factorial*

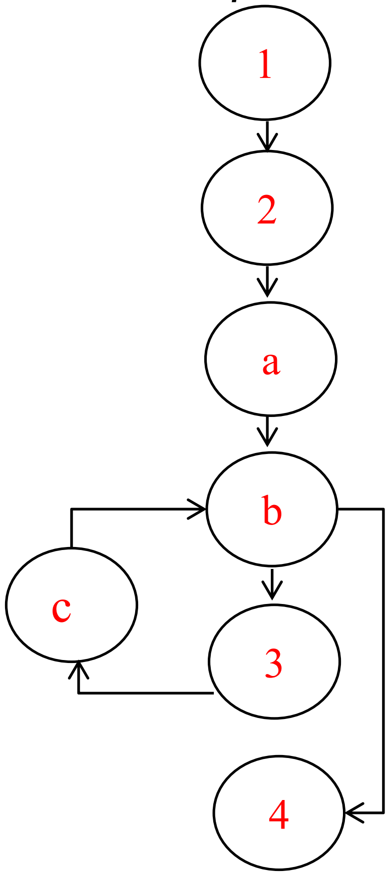

Fig. 4.7: CFG for factorial

There are 2 linearly independent paths. These are as:

1-2-a-b-4
1-2-a-b-3-c-b-4

Range of ‘terms’=0 to 6
Range of ‘x’=1 to 10

There are 2 test cases obtained from the path based testing which are as:

When terms =5 and x=7 (path will be covered as 1-2-a-b-3-c-b-4):

Expected output=24
Obtained output=24
So difference=| Expected output- Obtained output|=|24-24|=0

When terms =-2 and x=77 (path will be covered as 1-2-a-b-4):

Expected output=1
Obtained output=1
So difference=| Expected output- Obtained output|=0

Considering the tolerance allowed i.e $\alpha = 0.9$

$R_{log} = 1 - 1/2\{(0+0)/ 0.9\} = 1 - 0$

$= 1$

Hence, the probability of logical correctness obtained for the factorial function is 1.

Source Code for *polynomial_series*

```
double polynomial_series (int terms,long x)
{
        1.  int i;
        2.  double pow,fact, division,add=0.0;
            for(a i=0; b i<terms; c i++)
            {
        3.  pow = power(x,i);
        4.  fact = factorial(i);
        5.  division=pow/fact;
        6.  add=add+division;
            }
        7.  return add;
}
```

CFG for *polynomial_series*

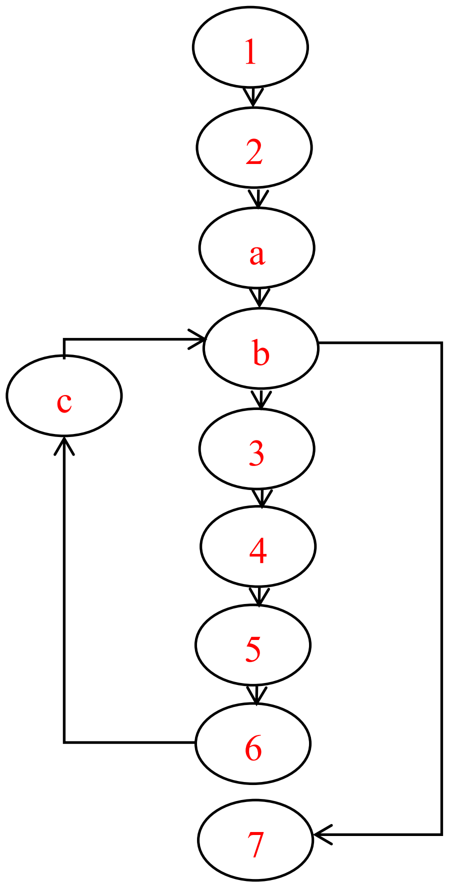

Fig. 4.8: CFG for *polynomial_series*

The calculation details of the glue code reliability values for the glue code used in the *polynomial_series* is present in Appendix A.1.

Table 4.1 Reliability values of components used in *polynomial_series*

Component	Operational correctness	Logical correctness	Total reliability
power	0.914	1	0.914
factorial	1	1	1

CDG for the *polynomial_series* is shown in Fig. 4.9:

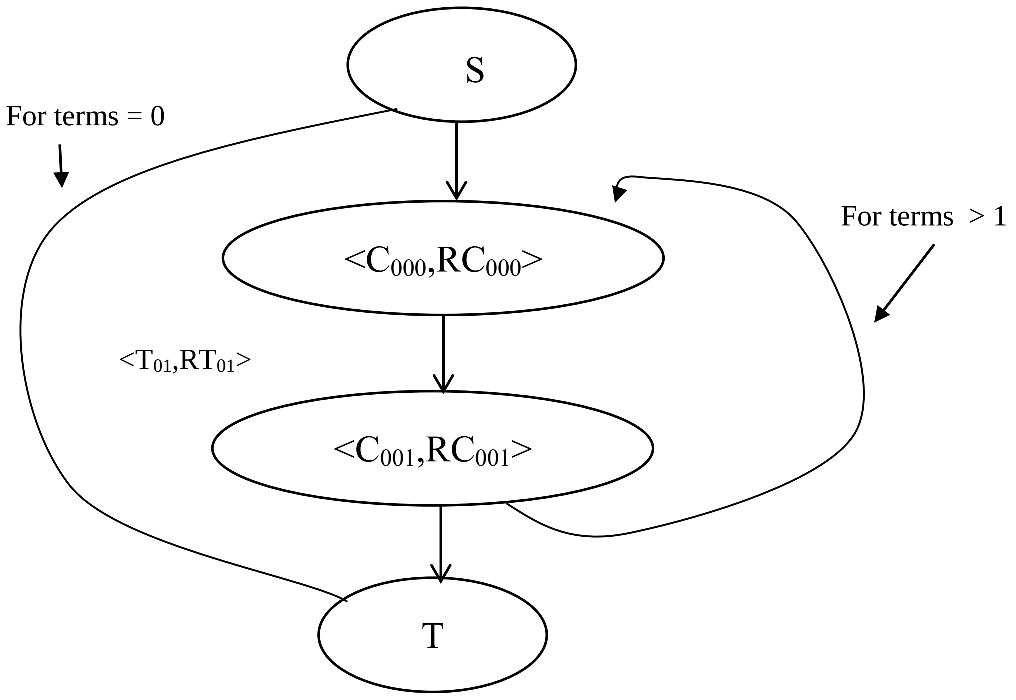

Fig.4.9: CDG for the *polynomial_series*

where

C_{000} represents *power* function,

C_{001} represents *factorial* function,

RC_{000} represents Reliability of the *power* function,

RC_{001} represents Reliability of the *factorial* function,

S corresponds to the start node,

T corresponds to the terminal node.

Reliability of execution scenario for *polynomial_series* using conditional probability matrix at level 2.

The **Conditional Probability Matrix** used at level 2 is

$$\begin{array}{c|ccc} & C & \mathbf{C_{000}} & \mathbf{C_{001}} \\ \hline \mathbf{C_{000}} & 0.914 & 0 & 1 \\ \mathbf{C_{001}} & 0 & 1 & 0 \end{array}$$

Here **C** corresponds to a conceptual ideal component assumed to be present in the execution scenario before the first component of the execution scenario.

where

C₀₀₀ represents *power* component,

C₀₀₁ represents *factorial* component.

Table 4.2 Reliability of various execution scenario based on number of terms:

Execution scenario	**Terms**	**(Reliability of execution scenario* glue code reliability of that scenario)**	**Total Reliability**
1st	0	1	1
2nd	1	$(0.914*1)^1$	0.914
3rd	2	$(0.914*1)^2$	0.835
4th	3	$(0.914*1)^3$	0.764
5th	4	$(0.914*0.3)^4$	0.006
6th	5	$(0.914*0.3)^5$	0.002
7th	6	$(0.914*0.3)^6$	0.000

Total average reliability of *polynomial_series*

$$= \left(\sum_{k=0}^{6} (Reliability\ of\ \ execution\ scenario\ \boldsymbol{k}\ polynomial\_series) * \ p(occurrence\ of\ execution\ scenario\ \boldsymbol{k}\ of\ polynomial\_series)\right) * p(task\ will\ be\ completed\ within\ deadline)$$

Total average reliability of *polynomial_series*

=(1*0.143+0.914*0.143+0.835*0.143+0.764*0.143+0.006*0.143+0.002*0.143+0.000*0.143)*0.509

= 0.503*0.509 = 0.256

4.3.2. Calculation of reliability value for *expression*

Since *expression* is an elementary component so *algorithm elem_Rel* is used to calculate its reliability.

Calculation of the probability of operational correctness of expression

For ***expression*** component:

Range for ‘b’ is 1-45,000

Range for ‘a’ is 1-23,000

Range for ‘c’ is 1-23,000

Out of range data for ‘a’, ‘b’, ‘c’=0

Probability of occurrence of data within range=1-0=1

So probability of operational correctness for *expression* is 1.

Calculation of probability of logical correctness for expression

Calculation of probability of logical correctness for ***expression*** component is as follows:

Source code for *expression*

```
double expression(long int b,long int a,long int c)
{
        1. long int sub,square,mul;
        2. double division;
        3. square=b*b;
        4. mul=4*a*c;
        5. sub=square-mul;
        6. division=sub/2;
        7. return division;
}
```

CFG for *expression*

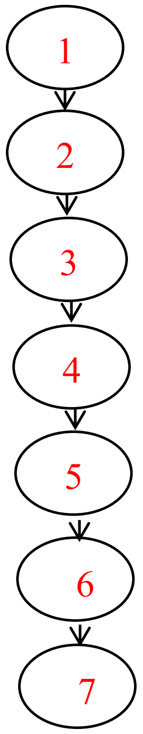

Fig.4.10: CFG for expression

There will be 1 linearly independent path:

1-2-3-4-5-6-7

Based on it, 1 test case is used to cover it.

Range for 'b' is = 1 to 45000

Range for 'a' is=1 to 23000

Range for 'c' is=1 to 23000

Considering these range, test case can be (b, a, c)=(5000,800,544)

Then,

Expected Output = 11629600

Obtained Output is =11629600

|Difference| =| Expected Output- Obtained output| =11629600-11629600|= 0

Considering the tolerance allowed i.e α= 0.9

R_{log} = 1- 1/1{(0)/ 0.9} =1- 0

= 1

So reliability of elementary component (i.e. *expression)* is = probability of operational correctness for *expression* * probability of logical correctness for *expression*

=1*1=1

Since *expression* is a component at level 1 so the probability of task completed within deadline is also multiplied with its reliability value (in accordance with the proposed approach).

Total average reliability of e*xpression* is = *Reliability of expression * p(task will be completed within deadline)*

=1*0.491=0.491

Table 4.3 Average reliability values of various components used in *main*

Component	Reliability value
polynomial_series	0.256
expression	0.491

Source Code for *main*

```
void main()
{
    1.  int i,terms,k;
    2.  double d,root1,root2,fact,pow,sum=0.0;
    3.  long int,x,b,a,c;
    4.  printf("Enter 1 for polynomial series or 2 for expression");
    5.  scanf("%d",&k);
    6.  if(k==1)
        {
    7.  printf("Enter terms and x values");
    8.  scanf("%d%ld",&terms,&x);
    9.  sum=polynomial_series(terms,x);
    10. printf("\n\n Sum of the series is :\n %f",sum);
        }
    11. else if(k==2)
        {
    12. printf("Enter values for b ,a and c");
    13. scanf("%ld%ld%ld",&b,&a,&c);
    14. d=expression(b,a,c);
    15. printf("\n Expression value is %f", d);
    16. if(d<0)
    17. printf("\n Roots are not real");
    18. else {
    19. root1 = (-b + sqrt(2*d))/(2*a);
    20.  root2 = (-b - sqrt(2*d))/(2*a);
    21. printf("\n Real roots are %f, %f",root1,root2);}
        }
    22. return 0;
```

}

The glue code reliability value obtained for scenario 1 and for scenario 2 of *main* are found to be 1 (Calculation details are present in Appendix A.2).

Average reliability of execution scenario1 for *Main* (including glue code)

= Average reliability of execution scenario1 for *Main * Glue code reliability for execution scenario 1* of *Main*

= 0.256*1=0.256

Average reliability of execution scenario 2 for *Main* (including glue code)

= Average reliability of execution scenario 2 for *Main * Glue code reliability for execution scenario 2 of Main*

=0.491*1=0.491

Average reliability of main = *Average* r*eliability of Execution Scenario1* for *Main* (including glue code) * *p(occ. of Scenario 1 in main)*Average Reliability of Scenario 2* for *Main* (including glue code) * *p(occ. of Scenario 2 in main)*

= 0.256*0.5+0.491*0.5=0.128+0.246=0.374

Hence the average reliability of the application is found to be 0.374.

The next section discusses the results obtained from the proposed approach and compares it with the results obtained from Yacoub et al. [21] approach.

5. RESULTS AND DISCUSSION

Failures in real time systems can arise from both functional errors as well as timing bugs. Hence, it is necessary to ensure both temporal correctness of programs used in real time applications in addition to provide functional correctness. Since, reliability value varies with the values of deadlines, hence reliability values of the application are computed for various deadline values for the same application (considered in section 4.3) which calculates either the roots of a quadratic equation or the value of the polynomial depending upon the user's choice.

5.1 Reliability values for various deadlines

Based on deadline value, the corresponding reliability value (shown in table 5.1) is obtained.

Table 5.1 The deadline and reliability value for the considered application

S.No.	**Deadline**	**Reliability**
1	1	0.174
2	2	0.367
3	3	0.37
4	4	0.372
5	5	0.374
6	6	0.374
7	7	0.374

The plot obtained between deadline and reliability value for the considered application is shown in Fig 5.1.

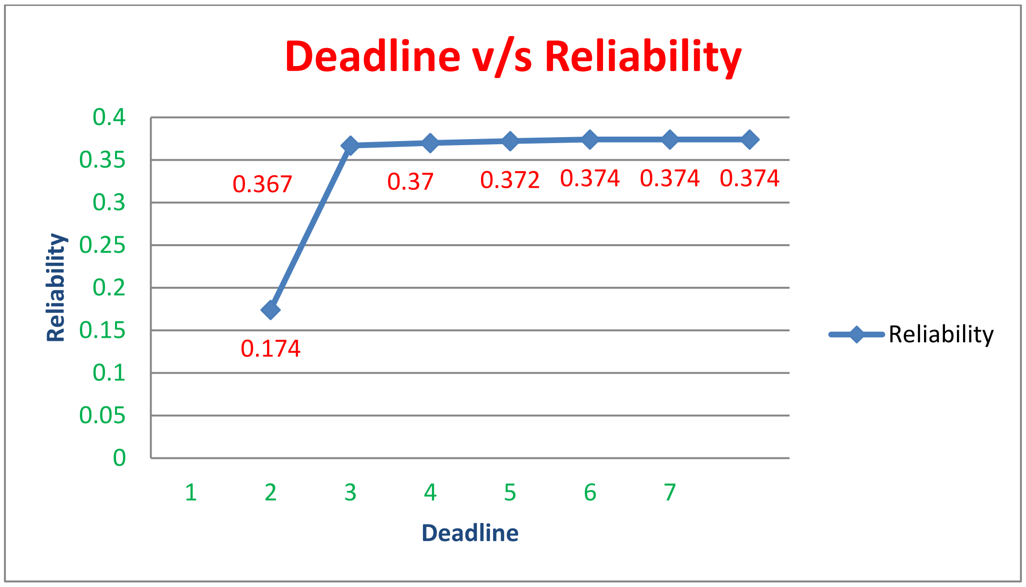

Fig. 5.1:The obtained plot between deadline and reliability value for the considered application.

From the plot it is found that when the deadline increases the reliability value initially increases drastically and then gradually until it reaches a value beyond which there is no increase in the reliability value with the increase in deadline.

5.2. Comparison with Yacoub et al. model

For real time applications, although very few work has been done, yet none of them is concerned with the development of a reliability assessment approach by considering the components dependencies, their failure dependencies based on the architecture of the system for various execution scenarios with respect to a given operational profile in the presence of timing bugs. Timing bugs are an important issue in real time systems, but is not handled explicitly by any of the existing approaches for reliability assessment. As for real time systems, many times it becomes a necessity for a given service to be delivered within the specified time deadline. Although, Yacoub et al.[21] model do not consider the deadline, yet it is taken into consideration for comparison as it does the assessment by considering the components dependencies, their failure dependencies based on the architecture of the system for various execution scenarios with respect to a given operational profile.

Here, once again the same application (considered in section 4.3) which calculates either the roots of a quadratic equation or the value of the polynomial depending upon the user's choice is taken into consideration. The reliability values for different scenarios are calculated using Yacoub et al. [21]. It is assumed that for a given/specified average execution time the number of iterations in the loop of the CDG for *polynomial_series* function is 4. The average reliability value obtained for the *polynomial_series* in accordance with Yacoub et al. is 0.338. Hence the average reliability value of the composite component C_{00} according to the model is 0.338. Reliability value of elementary component C_{01} is 1. Thus, the average reliability value of the application obtained from Yacoub et al. is 0.669. This value is greater than the average reliability value of the proposed approach (where deadline has been taken into consideration).

Table 5.2 Comparison with Yacoub et al. model

Approach	Avg. reliability
Proposed approach	0.374
Yacoub et.al.	0.669

Hence, it can be seen that the proposed approach takes into account several factors explicitly for real time applications like timing bugs, etc. Moreover, in many of the component based approaches (with the exception of few) it is assumed that the reliability values of the elementary components are either known or taken from elsewhere (like third party vendors, etc.); since there is no guideline to obtain the reliability values of the elementary components. The proposed approach is not only able to provide the average reliability value for the given application, but is also able to obtain the reliability values for various components (both elementary and composite), without using the values provided by the third party vendors.

The next section concludes with the discussion on the advantages and limitations of the existing approach and provides guidelines for future enhancements.

6. CONCLUSION

Real time systems are those systems which must guarantee to response correctly within strict time constraint or within deadline like satellite systems, traffic control systems, etc. The severity of consequences resulting from the failure of these systems is usually is very high. Hence, the reliability should be estimated accurately in order to minimize or avoid the losses resulting from the failure of such systems. Failure of real time systems which are used in real life applications can arise from both functional error as well as timing bugs. Therefore, it is necessary to provide temporal correctness of embedded program used in real time applications in addition to providing functional correctness. It has been observed that many approaches do not consider dependency among components, timing bugs, failure dependency and various execution scenarios possible for a given operational profile in the presence of indefinite loops and other instance characteristics. Also, for real time systems, many times it becomes a necessity for a given service to be delivered within the specified time deadline, but many approaches do not explicitly take this issue into consideration.

The proposed algorithm which considers the components dependencies, their failure dependencies (based on the architecture of the system) for various execution scenarios with respect to a given operational profile in the presence of timing bugs, provides a versatile and cost effective approach for the reliability assessment of real time software applications. The developers and integrators can readily use this approach to assess the reliability of their applications without relying on the reliability values provided by other third party quality assurers or testers for any component. As, there can be several issues like the third party vendors may be biased, the testing environment may not be the same as the operational environment, etc. Hence, it is difficult to ensure that the reliability values obtained from these third party vendors are correct and effective for the given application. Therefore, the proposed approach serves to be easy and cost effective. Currently, the limitation of this work is that it takes a considerable amount of time to do the assessment manually even for small input domains. Hence, work can be done in future to automate the process of the reliability assessment using the proposed approach so that results can be obtained without much effort and time. Thus, considering various aspects it can be said that the proposed approach closely captures the behavior of the real time software application as it considers various aspects like the components dependencies, their failure dependencies (based on the architecture of the system) for various execution scenarios with respect to a given operational profile in the presence of timing bugs. Therefore, it serves to be a useful approach for the reliability assessment of real time applications.

REFERENCES

[1] http://users.ece.cmu.edu/~koopman/des_s99/sw_reliability

[2] http://universe.bits-pilani.ac.in/uploads/rts-intro-slides.pdf

[3] https://www.ima.umn.edu/~arnold/disasters/ariane5rep.html

[4] A. L. Goel."An analysis of competing software reliability models*", IEEE Transactions on Software Engineering*, vol.6,pp.501-502,1980.

[5] A.P.Singh and P.Tomar."A new model for reliability estimation of component based software", *Advance Computing Conference,* 2013, pp.1431-1436.

[6] B. Littlewood. "Software reliability model for modular program structure",*IEEE Transaction. Reliability*, vol.28, pp.241-246 ,1979.

[7] C.J. Hsu and C.Y. Huang. "An adaptive reliability analysis using path testing for complex component-based software systems", *IEEE Transactions on Reliability,*vol. 60, pp.158-170,2011.

[8] C.Huang and M.R.Lyu. "A unified scheme of some nonhomogenous poisson process models for software reliability estimation", *IEEE transactions on Software Engineering*, vol. 29, pp.261-269,2003.

[9] G. Pai. "A Survey of Software Reliability Models", *arXiv preprint arXiv*,2013,pp.1304-4539.

[10]H.Kopetz.*Real-Time Systems, Design Principles for Distributed Embedded Applications*. Springer Science & Business Media, 2011.

[11] H.Pham, M.Pham."Software Reliability Models for Critical Applications", *Idaho National Engineering Laboratory EG&G Idaho*, 1991.

[12] K.G. Popstojanova ,K.S. Trivedi."Architecture-based approach to reliability assessment of software systems", *Performance Evaluation*,vol.45, pp.179-204,2001.

[13] L.Nautiyal, Dr. N. Gupta and Dr. S.C.Dimri. "A new path based reliability approach for estimation of reliability of Component Based Software Development", *International Journal of Computer Science Engineering*,vol.1,pp.295-299,2013.

[14] M.Xiaoguang, D.Yongjin. "A General Model for Component-Based Software Reliability", *Proceedings of the 29th EUROMICRO Conference*,2003, pp. 395-398.

[15] R. C. Cheung."A user-oriented software reliability model*", IEEE Transactions on Software Engineering*,vol.6, pp.118-125,1980.

[16] R.Mohd.,M. Nazir.“ Software Reliability Growth Models: Overview and Applications”, *Journal of Emerging Trends in Computing and Information Sciences*, vol.3,pp.1309-1320,2012.

[17] S.Biswas, R.Mall and M.Satpathy. “Task dependency analysis for regression test selection of embedded programs.” *Embedded Systems Letters, IEEE* ,vol.4 ,pp.117-120 ,2011.

[18] S. Gokhale. “Accurate Reliability Prediction based on Software Structure”, *Proc. of IASTED Conference on Software Engineering and Applications*,2003,pp.23-25.

[19] S. Gokhale, W.E. Wong, K. Trivedi and J.R. Horgan.“An analytical approach to architecture based software reliability prediction”, *Proceedings of the Third International Computer Performance and Dependability Symposium*,1998, pp.13-22.

[20] S N. Weiss. “An Extended Domain-Based Model of Software Reliability*”,IEEE Transactions on Software Engineering,* vol. 14, pp.1512-1524,1988.

[21] S.Yacoub.“Scenario Based Reliability Analysis of Component Based Software”, *IEEE Transactions on Reliability*, vol.53, pp. 22-31,2004.

[22] T. Gayen.“Analysis and proposition of error based model to predict the minimum reliability of software”,*International Conference on Education Technology and Computer, 2009,* pp.40-44.

[23] T.Gayen and R.B.Misra. “Reliability Assessment of Elementary COTS Software Component”, *International Journal of Recent Trends in Engineering*, vol.1, pp.196-200,2009.

[24] T.Gayen and R.B.Misra.“Reliability bounds prediction of COTS component based software application*”, International Journal of Computer Science and Network Security*,vol.8, pp.219-228,2008.

[25] V. Goswami and Y.B. Acharya. “Method for Reliability Estimation of COTS Components based Software Systems”, *Proceedings of 20th International Symposium on Software Reliability Engineering, ISSRE*, 2009,pp.123-129.

[26] W. Everett.“Software component reliability analysis”, *Proceedings of the Symposium on Application-specific Systems and Software Engineering Technology*, 1999, pp. 204-211.

APPENDIX

A.1 Test code for *polynomial_series*

CFG for the test code of *polynomial_series*

```
double polynomial_series_Test (int terms,long x)
{
    1.  int i;
    2.  double pow,fact,division,add=0.0;
    3.  long int  power[20]={1,7,49,343,2401,16807};
    4.  long int factorial[20]={1,1,2,6,24,120};
        for(a i=0; b i<terms; c i++)
        {
    5.  pow = power[i];
    6.  fact = factorial[i];
    7.  division=pow/fact;
    8.  add=add+division;
        }
    9.  return add;
}
```

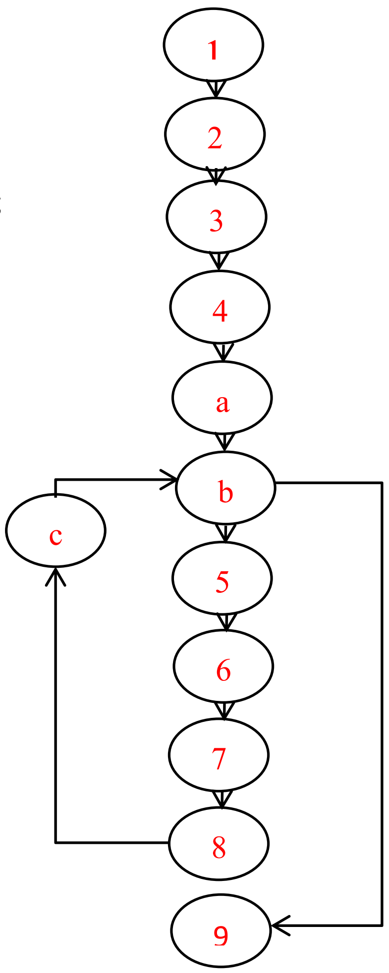

Fig.A.1:CFG for the test code of *polynomial_series*

Calculation of the probability of operational correctness for glue code in *polynomial_series*

Range for 'terms'= 0 to 6

Range for 'x'=1 to 10

For terms = 0, the glue code inside the loop is not executed.

For terms = 1,2,3 there is no overflow error, hence the probability of operational correctness = 1.

For terms = 4,5,6 there is overflow error when x = 1,2,4,5,7,8,10.

Therefore, the probability of occurrence of overflow errors = 7/10 = 0.7

The probability of non-occurrence of overflow errors = 1- 0.7 = 0.3

Hence, the probability of operational correctness = 0.3

Calculation of the probability of logical correctness for glue code in p*olynomial_series*

In CFG, there are 2 linearly independent paths which are as:

1-2-a-b-7

1-2-a-b-3-4-5-6-c-b-7

Thus there are 2 test cases used to cover all linearly independent paths. These are as:

When terms= -99 and x=6, path is covered as 1-2-a-b-7:

Expected output=0

Obtained output=0

So difference=| Expected output- Obtained output|=0

When terms =5 and x=7, path is covered as 1-2-a-b-3-4-5-6-c-b-7:

Expected output = 189.708

Obtained output = 189.708

|Difference |= |189.708-189.708|=0

Considering the tolerance allowed i.e α= 0.9

R_{log}= 1- 1/2{(0+0)/ 0.9} =1-0=1

Hence, the probability of logical correctness obtained for the glue code present in *polynomial_series* is 1.

Table A.1 Glue code reliability values for different values of terms

Terms	Glue code reliability
1	1
2	1
3	1
4	0.3
5	0.3
6	0.3

A.2 Test code for *main*

```
void main_Test()
{
1.      int i,terms,k;
2.      double d, root1,root2,sum=0.0;
3.      long int fact,pow,x,b,a,c;
4.      printf("Enter 1 for polynomial series or 2 for expression");
5.      scanf("%d",&k);
6.      if(k==1)
        {
7.      printf("Enter terms and x values");
8.      scanf("%d%ld",&terms,&x);
9.      sum=189.708;
10.     printf("\n\n Sum of the series is :\n %f",sum);
        }
11.     else if(k==2){
12.     printf("Enter values for b ,a and c");
13.     scanf("%ld%ld%ld",&b,&a,&c);
14.     d=11629600;
15.     printf("\n Expression value is %f",d);
16.     if(d<0)
17.     printf("\n Roots are not real");
18.      else {
19.     root1 = (-b + sqrt(2*d))/(2*a);
20.     root2 = (-b - sqrt(2*d))/(2*a);
21.     printf("\n Real roots are %f, %f",root1,root2);}
```

```
                        }
        22.             return 0;
}
```
CFG for the test code of *main*

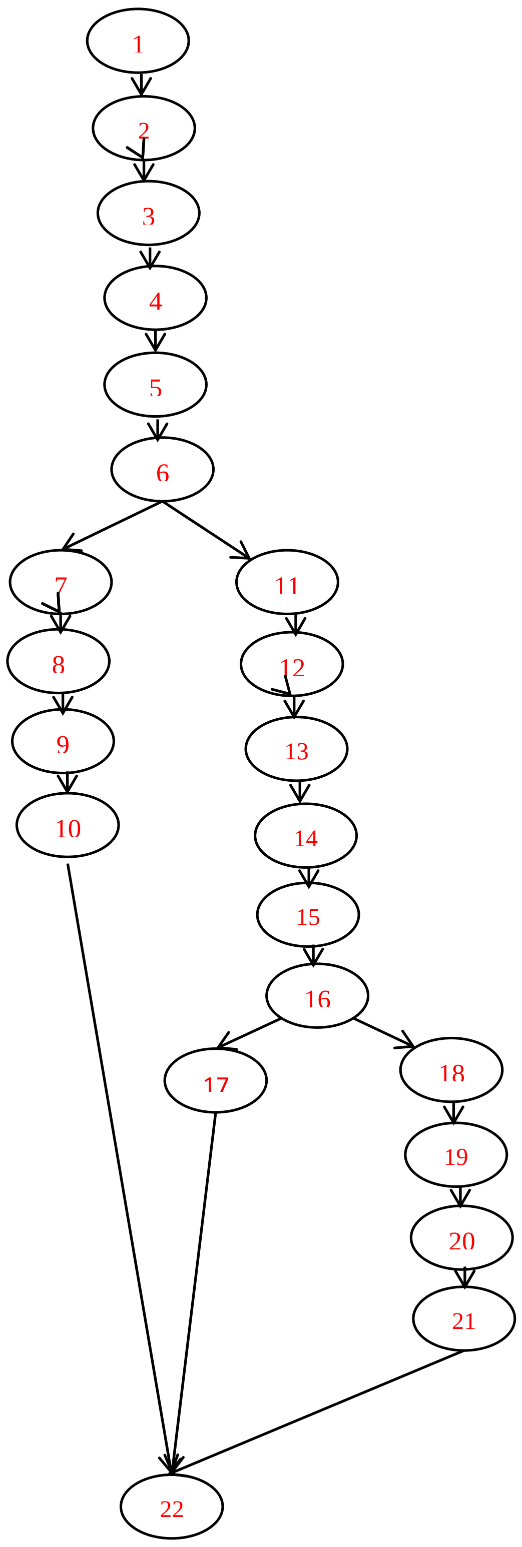

Fig.A.2: CFG for the test code of *main*

Calculation of the probability of operational correctness for *main*

The test code for *main* is found to execute without any operational error. Hence, the probability of operational correctness is found to be 1.

Calculation of the probability of logical correctness for *main*

Since in *main* there are 2 scenarios.
Using path based testing, there exists 3 paths so 3 test cases are used to cover it.
The test case taken for Scenario 1 is (5,7) and for Scenario 2 is (200,250,400)and (5000,800,544)
The logical correctness is as:

When 'k'= 1, path will be covered as 1-2-3-4-5-6-7-8-9-10-22so test case for this path is (terms, x) = (5, 7).
Scenario 1 will be covered.

Expected output=189.708
Obtained output=189.708
So difference=| Expected output- Obtained output|=0

When 'k'=2, and d<0 path will be covered as 1-2-3-4-5-6-11-12-13-14-15-16-17-22) so test case for covering this path is (b, a, c) = (200,250,400).
Scenario 2 will be covered.

Expected value = Roots are not real
Obtained value=Roots are not real
Difference=|Expected value-Obtained value|= 0

When 'k'=2, and d>=0 path will be covered as 1-2-3-4-5-6-11-12-13-14-15-16-18-19-20-21-22) so test case for covering this path is (b, a, c) = (5000,800,544).
Scenario 2 will be covered.

For root 1,

Expected value = -0.11

Obtained value=-0.11

Difference=Expected value-Obtained value= -0.11+0.11=0

For root 2,

Expected value =-6.14

Obtained value=-6.14

Difference=Expected value-Obtained value=-6.14+6.14=0

So total difference =0

Considering the tolerance allowed i.e α= 0.9

R_{log} = 1- 1/3{(0+0+0)/ 0.9} =1

= 1

Hence the reliability of the glue code in main is found to be 1*1 = 1.