Limiting the Reliability of Component-Based Software System

Neha Garg
Graphic Era University, Dehradun, India
nehagarg.february@gmail.com

Lata Nautiyal
Graphic Era University, Dehradun, India
Email: lata.nautiyal1903@gmail.com,

Preeti
Graphic Era University, Dehradun, India
Email: preetishivach2009@gmail.com

Abstract— Component-Based Software Development has proved itself the best among all the software development techniques to deliver the efficient, timely and reliable software product. The burning issue of loss of control casts shadow on the advantageous face of the component-based development. Deep analysis of the Reliability has been a substantial track of the safety management and involved high importance. It is highly essential aspects to software systems reliability as decision makers are mostly concerned in estimating the future occurrences of failures of the software system. Thus the goal of this article is to provide a solution to the developers and integrators to reclaim some control of their development process by forecasting the upper and lower bound on the reliability. Proposed approach introduces component dependency graphs and reliability estimation is done on the basis of execution paths of the component-based software.

Keywords—Component, component-based software system, reliability limit, component dependency graph, execution path.

I. INTRODUCTION

In recent times Component-Based Software Development (CBSD) has proved itself the best among all the software development techniques to deliver the efficient, timely and reliable software product [1]. As day by day consumer products are becoming highly software intensive, the needs of generating and maintaining software products is also increasing in rapid growth. As it is well known fact that software productivity can be tremendously maximize with reuse of the software product because reused software components need not to be developed from scratch. The foremost merits associated with component-based technologies include: development of condensed system, quick installation, reduced cost, enhanced quality, and condensed system evolution and less maintenance cost [2, and 3].

Component-based Software Systems (CBSS) are centered and focused on the notion of lump together various independent and pre-defined components which may have different design, code and frame architecture [4]. The trustworthiness of these assembled components affects the trustworthiness of the entire software system [5]. Non-functional requirements like fault tolerance, independency, interoperability, safety, maintainability, confidentiality, quality, and reliability for the software products are an important part of physical as well as logical products [6, 7, 8, and 9].

System reliability can also be defined as the probability that system will accomplish its intended function for an explicit period of time under certain conditions [10]. Concentrating on safety, reliability analysis aims at the quantification of the probability of failure of the entire software system [11, and 12]. It is highly important aspect to software systems reliability, as decision makers are mostly concerned in estimating the future occurrences of failures of the software system. The burning issue of loss of control casts shadow on the advantageous face of the CBSD. Therefore the goal of this article is to help developers and integrators to reclaim some control of their CBSS by forecasting the upper and lower bound on the reliability.

II. RELATED WORK

Component-Based Software Engineering is an advanced approach that takes some pre-defined, tested and proved components; line up them to be integrated with each other; customize them so that fully functional and reliable software can be produced [13]. The construction of reliable CBSSs requires number of approaches to aid the software developers in ensuring that the software architecture, selected components and finally the
constructed software system meet the preferred excellence requirements. Numerous analytical methods have been developed for system-level reliability prediction.

The approach presented by Marko Palviainen et al. [14] reports software reliability estimation throughout the design and implementation phases; it offers an innovative method by combining both expected and measured reliability values with heuristic estimates in order to facilitate a smooth reliability evaluation process. This method contributes by integrating the component-level and the system-level reliability prediction activity to support the incremental and iterative development of reliable CBSSs.

Fan Zhang et al. [15] proposed architecture-based reliability evaluation process which reflects the theory of fault propagation. Architecture-based reliability analysis can be achieved as early as the design phase of the software application. With architecture-based software reliability analysis, we can predict the relationship between overall software reliability and the reliability of the individual components.

Gokhle et al. [16] converse the flexibility that discrete-event simulation offers for deeply studying the component-based applications. This procedure adopts that the application has a control flow graph. The simulation uses component failure and repair rates to simulate failures while executing the application. The total number of failures is calculated for the application under simulation, and its reliability is estimated. The simulation assumes continual execution times and totally overlooks the failures of component interfaces and links.

Wang and Hang [17] proposed a technique called Reliability Analysis based on Rewrite Logic (RABRL). This technique is based on analysis of operational profiles and specifications. These specifications are implemented one by one by using the rewrite language Maude. Transition probabilities and the expected number of components which will be stayed are statistically analyzed by the execution progression.

Lo [18] also proposed a software reliability estimation model based on a Support Vector Machine (SVM) and Genetic Algorithm (GA). This model postulates that the current failure data alone are adequate for estimating reliability. Reliability estimation parameters for the SVM are determined by the GA. This model is less dependent on failure data than are other models.

Goswami and Acharya [19] proposed an approach which considers the component usage ratio in reliability estimation process of software. The component usage ratio is computed through mathematical formulas. This approach may be used in real-time applications due to the flexibility of the component usage ratio.

Yacoub et al. [20] proposed an approach called Scenario-based Reliability Analysis for estimating the reliability. This approach introduces component dependency graphs that can be extended for complex distributed systems. Using this algorithm, sensitivity may also be analyzed as a function of component reliabilities and link reliabilities. The approach is based on scenarios which can be captured with sequence diagrams, which means that this approach can be automated. A major disadvantage of the given method is that it does not reflect error propagation among the components.

Gayen et al. [21] proposed that the reliability of software product is composed of commercial off-the-shelf (COTS) components is extremely reliant on component reliability. Goseva-Popstojanova et al. [22] classified architecture-based approaches to reliability assessment of CBSSs into three classes. These classes are:

- **State-based approaches:**
  In these approaches software architectures and failure behaviors are represented as Markov chains or semi-Markov processes,

- **Path-based approaches:**
  In path based approach reliability is estimated for sets of execution scenarios [23, and 24], and

- **Additive models:**
  Additive models focus on estimating the time-dependent failure intensity of the system using failure data for the components.

Everett [25] described six steps for conducting an analysis of software component reliability.

1) Divide the software into components,
2) Characterize the properties of each component,
3) Define the usage of each component,
4) Model the reliability of each component,
5) Superimpose the component reliabilities, and
6) Perform a confirmatory analysis through testing.

Numerous approaches, such as that in Whittaker [26], use Markov chains to successfully develop the reliability models (Markov Models, MMs) for software product. MMs are used to capture system states and the transitions from one state to another, where the transitions are the result of component failures.
On the basis of this state of art, we conclude that reliability is a real-world spectacle and that in CBSS there should be some limitations on the reliability value. In our proposed model, we assume that CBSS reliability may be expressed in terms of the reliability of components.

### III. LIMITING THE RELIABILITY

The algorithm given below depicts the process followed for obtaining the bounds on the reliability of a CBSS:

1. Draw the Component Dependency Graph of the software (Section 3.1),
2. Reduce the CDG (Section 3.2),
3. Identify the possible execution paths (Section 3.2),
4. Evaluate the reliability of the software for each execution path (Section 3.3),
5. The upper bound on reliability is the maximum value obtained in step 4,
6. The lower bound on reliability is the minimum value obtained in step 4.

#### 3.1 Component Dependency Graph

Beginning the process with the basic thought of control flow graphs, we develop a model named component dependency graph. Control flow graphs are the traditional method of revealing the structure, decision points, and branches in program code [20, and 27]. A control flow graph is a directed graph that consists of a set of nodes and directed edges $G = <N, E>$. Each node represents one or more program statements. $N$ is the total number of nodes in $G$. Each edge represents the transfer of execution control from source to destination. Each edge is an ordered-pair $<N_i, N_j>$.

The ic theory of the control flow graph are adapted to component-based applications to symbolize the dependency between components and all the potential execution paths. We call this graph *Component Dependency Graph* (CDG). In this section we define the graph for a component based application.

**Definition 1:** Component Dependency Graph "CDG"

A component dependency graph is defined by $CDG = <N, E, s, t>$, where:

- $<N, E>$ is a directed graph,
- $s$ is the start node,
- $t$ is a termination node,
- $N$ is a set of nodes in the graph, $N = \{n\}$, and
- $E$ is set of edges in the graph, $E = \{e\}$.

**Definition 2: A Node "n"**

A node $n$ models a component and is defined by the tuple $<C_i, R(C_i)>$ where:

- $C_i$ is the component name,
- $R(C_i)$ is the component reliability.

**Definition 3: Component Reliability "R(C_i)"**

It is the probability that the component $C_i$ will execute correctly (fault free) during its course of execution.

**Definition 4: A Directed Edge "e"**

A directed edge $e$ models the execution path from one component to another and is defined by the tuple $<T_{ij}>$ where:

- $T_{ij}$ is the transition name from node $N_i$ to $N_j$ and denoted $<N_i, N_j>$.

Figure 1 shows a CDG consisting four components:
3.2 Reducing Component Dependency Graph and Identifying Paths

When program or software executes it follows a particular path. The execution path followed by the program depends on the input passed by the user at the time of execution. There may be many execution paths of the software, so the execution paths consist of different components. As reviewed in literature that reliability of the CBSS is a function of reliabilities of the components integrated for the system. Reliability of following a path is different from the reliability when the system execution follows another path.

Rather than considering all decision outcomes within CDG independently, we will focus on the decision outcomes that are involved with component calls. The design reduction technique helps identify those decision outcomes, so that it is possible to exercise them independently during proposed approach. The idea behind design reduction is to start with a CDG, remove all control structures that are not involved with component calls, and then use the resultant “reduced” flow graph to find the limits on reliability. Figure 2 shows a systematic set of rules for performing design reduction. The rules work together to eliminate the parts of the dependence graph that are not involved with component calls. The repetitive rule eliminates top-test loops that are not involved with component calls. The looping rule eliminates bottom test loops that are not involved with component calls. It is important to preserve the component’s connectivity when using the looping rule, since for poorly-structured code it may be hard to distinguish the “top” of the loop from the “bottom.” For the rule to
apply there must be a path from the component entry to the top of the loop and a path from the bottom of the
loop to the component exit.

Figure 3 shows a CDG before and after design reduction.

Figure 3. (a) CDG, (b) CDG after reduction

Here C1, C2, C3 and C4 represent components. The loops in CDG represent the repeated execution of the
component (*like during a repetitive call*) (C2 has a loop in figure 3) 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 figure 3). Next step is to identify the execution paths of CDG. From the CDG
the possible execution paths are separated out as follows (see figure 4):

### 3.1 Evaluating Reliability

A particular path will involve the activation of predefined components, which lie on the path taken during the
execution of the complete system [28]. Probability of following a particular path is evaluated as follows.

Let N be the total number of times the software process the data and produces output. And there are k possible
paths of execution. Out of N times, N₁, N₂, N₃... Nₖ times the i_th execution path is followed i.e. N₁ times the
execution follows path P₁, N₂ times the execution follows path P₂ and so on. Then the probability of following i_th
path (ρᵢ) will be:

\[ \rho_i = \frac{N_i}{N} \]

Where the value of path propagation probability is \( 0 < \rho_i < 1 \) and total of all paths’ probabilities is equal to 1,

\[ \sum_{i=1}^{k} \rho_i = 1 \]
The process of reliability evaluation is illustrated with an example. Suppose we have a CBSS having the CDG shown below (figure 5).

Figure 5. Component Dependency Graph
From figure 5, the possible execution paths are:

P₁  (C₁-C₂-C₃-C₄-C₉),
P₂  (C₁-C₅-C₆-C₉),
P₃  (C₁-C₂-C₃-C₆-C₉),
And
P₄  (C₁-C₇-C₈-C₉).

Let the length of simulation run be 1000, i.e. \(N=1000\), and out of \(N\), \(N_1\) times we get the output by following path \(P_1\), \(N_2\) times we get output by following path \(P_2\) and so on. For illustrating the proposed approach, we arbitrarily took \(N_1=250\), \(N_2=300\), \(N_3=300\) and \(N_4=150\), the probability of obtaining the output by following path \(P_1\) will be,

\[
\rho_1 = \frac{250}{1000} = 0.250,
\]

Similarly,

\[
\rho_2 = \frac{300}{1000} = 0.300,
\]

\[
\rho_3 = \frac{300}{1000} = 0.300,
\]

and

\[
\rho_4 = \frac{150}{1000} = 0.150.
\]

Let the failure probability of the components is uniform and it is 0.01, i.e. reliability of a component will be,

\[
\Phi_c = (1 - \text{failure probability}) = (1 - 0.01) = 0.99.
\]

Reliability of the system, when \(i^{th}\) (\(1 < i < n\), \(n\) is the number of possible paths) path is followed can be defined as:

\[
R(P_i) = \rho_i \sum_{j=1}^{n} \Phi_j,
\]

where \(j\) is the number of components between start and terminated node in the path \(P_i\).

The summation may be greater than 1. As known to all, the reliability of any system ranges from 0 to 1. So we have to convert \(R(P_i)\) to the range 0 to 1. For illustration suppose the summation is 3.99, we will use the expression

\[
=(100 - x) / 100
\]

\[i.e.
=(100 - 3.99) / 100
\]

\[=96.01 / 100
\]

\[=0.9601
\]

So calculating reliability of each path by using above formula;
\[ R(P_1) = \rho_1[R(C_1) + R(C_2) + R(C_3) + R(C_4) + R(C_5)] \]
\[ = 0.250[0.99 + 0.99 + 0.99 + 0.99 + 0.99] = 1.2375 \]
Converting to range,
\[ R(P_1) = (100 - 1.2375) / 100 = 0.987625 \]

\[ R(P_2) = \rho_2[R(C_1) + R(C_2) + R(C_3) + R(C_6) + R(C_9)] \]
\[ = 0.300[0.99 + 0.99 + 0.99 + 0.99 + 0.99] = 1.188 \]
Converting to range,
\[ R(P_2) = (100 - 1.188) / 100 = 0.98812 \]

\[ R(P_3) = \rho_3[R(C_1) + R(C_2) + R(C_3) + R(C_6) + R(C_9)] \]
\[ = 0.300[0.99 + 0.99 + 0.99 + 0.99 + 0.99] = 1.485 \]
Converting to range,
\[ R(P_3) = (100 - 1.485) / 100 = 0.98515 \]

\[ R(P_4) = \rho_4[R(C_1) + R(C_2) + R(C_7) + R(C_9)] \]
\[ = 0.150[0.99 + 0.99 + 0.99 + 0.99] = 0.594 \]
Converting to range,
\[ R(P_4) = (100 - 0.594) / 100 = 0.99406 \]
So the upper bound on the reliability of the system is 0.99406 i.e. 99.40% and the lower bound is 0.98515 i.e. 98.51%.

IV. CONCLUSION

Component-based software development plays the very important role in the field software development. The key benefits linked with component-based latest methodologies which include: development of innovative systems, quick installation, reduced cost, enhanced quality, and condensed system evolution and less maintenance cost. Reliability analysis has been a significant direction of safety management and attached great importance. It is highly essential and important to forecast system reliability as decision makers are generally concerned in estimating the future occurrences of system failures. This paper presents an innovative approach to limit the reliability of component-based software systems. As by knowing the upper and lower bound, one can easily predict the range of reliability values an application can have. We have used path-based approach to estimate the bounds on reliability of a component-based system. Reliability of a system is a function of reliability of the components integrated to form the system. In future, we can compare the proposed approach with other proposals. Also we don’t consider the error propagation probability and transition failure in estimating the reliability limits, so this could also be an extension to the proposed approach.

REFERENCES