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:

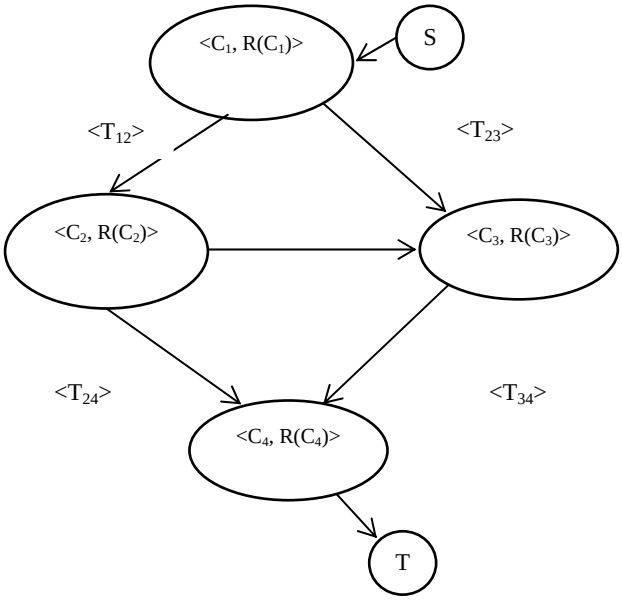

Figure 1. *A Sample Component Dependency Graph*

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.

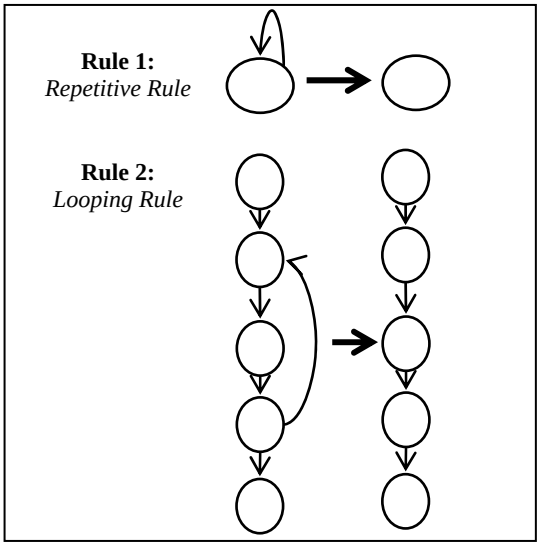

Figure 2. Rules for Reducing CDG

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.

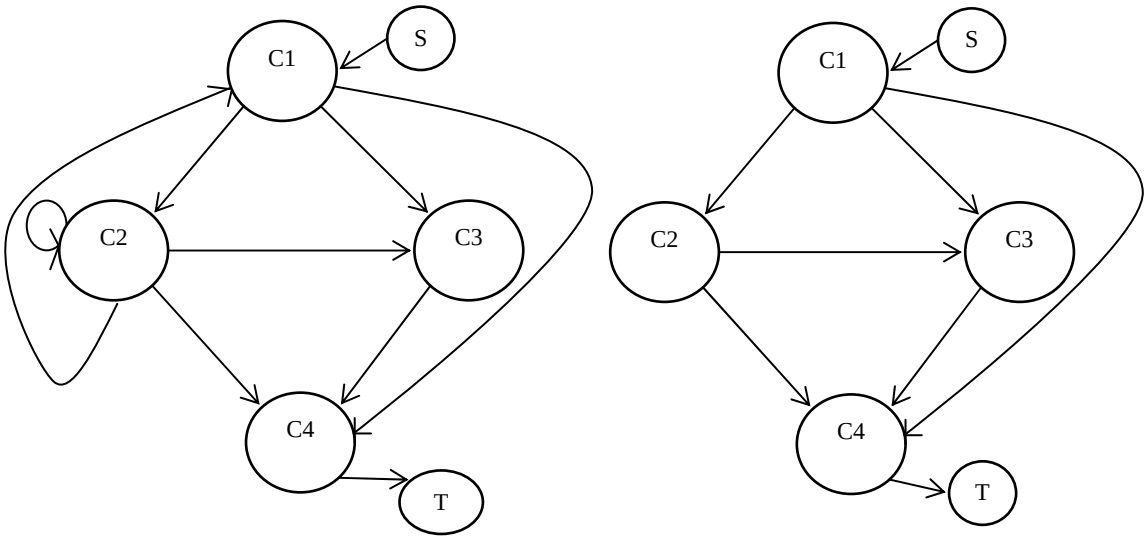

Figure 3. (a) CDG, (b) CDG after reduction

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, N_1, N_2, N_3… N_k times the ith execution path is followed i.e. N_1 times the execution follows path P_1, N_2 times the execution follows path P_2 and so on. Then the probability of following ith path (ρ_i) will be:

$$\rho_i = N_i / N$$

Where the value of path propagation probability is $0 < \rho_i < 1$ and total of all paths' probabilities is equal to 1, i.e.

$$\sum_{i=1}^{k} \rho_i = 1$$

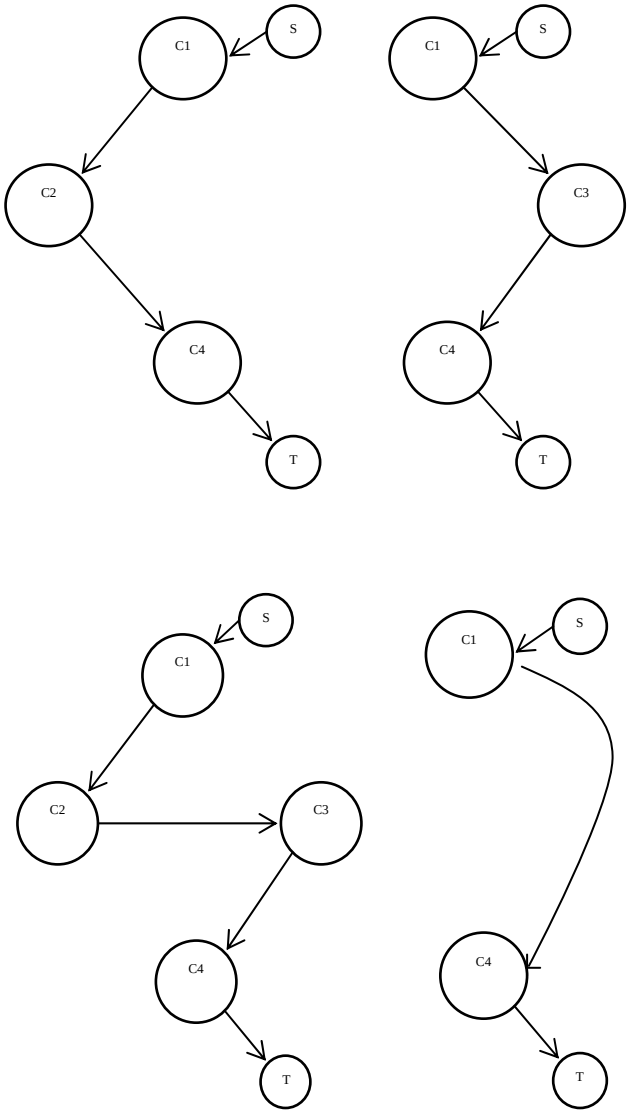

Figure 4. Possible Execution Paths

The process of reliability evaluation is illustrated with an example. Suppose we have a CBSS having the CDG shown below (figure 5).

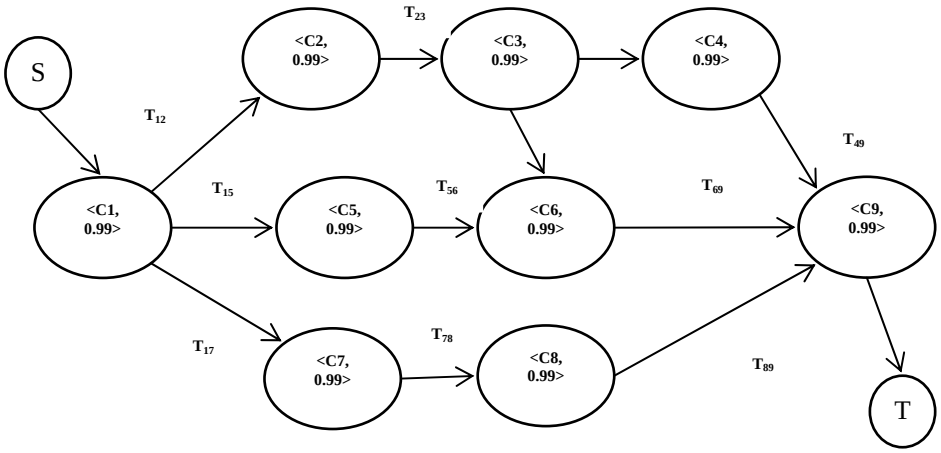

Figure 5. Component Dependency Graph

From figure 5, the possible execution paths are:

P_1 (C1-C2-C3-C4-C9),

P_2 (C1-C5-C6- C9),

P_3 (C1-C2-C3- C6-C9),

And

P_4 (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_3=150, the probability of obtaining the output by following path P_1 will be,

ρ_1 = 250/1000 =0.250,

Similarly,

ρ_2 = 300/1000=0.300,

ρ_3 = 300/1000=0.300,

and

ρ_4 = 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,

$\Phi_c = (1 - \textit{faliure probability})$ = (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;

$$R(P_1) = \rho_1[R(C_1) + R(C_2) + R(C_3) + R(C_4) + R(C_9)]$$
$$= 0.250[0.99 + 0.99 + 0.99 + 0.99 + 0.99] = 1.2375$$

Converting to range,

$$R(P_1) = (100 - 1.2375)/100 = 0.987625$$

$$R(P_2) = \rho_2[R(C_1) + R(C_5) + R(C_6) + R(C_9)]$$
$$= 0.300[0.99 + 0.99 + 0.99 + 0.99] = 1.188$$

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,

$$R(P_3) = (100 - 1.485)/100 = 0.98515$$

$$R(P_4) = \rho_4[R(C_1) + R(C_7) + R(C_8) + R(C_9)]$$
$$= 0.150[0.99 + 0.99 + 0.99 + 0.99] = 0.594$$

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

[1] Lata Nautiyal, Neena Gupta, and S. C. Dimri, "A Novel Approach to Component-Based Software Testing", ACM SIGSOFT Software Engineering Notes, Vol. 39, No. 6, pp. 1-4, Nov 2014.

[2] Lata Nautiyal, Neena Gupta, "ELICIT- A New Component-Based Software Development Model", International Journal of Computer Applications, Vol. 63, No. 21, pp. 53-57, Feb 2013.

[3] Lata Nautiyal, Neena Gupta, "ELITE Plus – Component Based Software Process Model", International Journal of Computer Applications, Vol. 90, No. 5, pp. 1-7, March 2014.

[4] Lata Nautiyal, Neena Gupta, and S. C. Dimri, "Measurement of Reliability of a Component-Based Development using a Path-Based Approach", ACM SIGSOFT Software Engineering Notes, Vol. 39, No. 6, pp. 1-4, Nov 2014.

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

[6] W. Farr, "Software reliability modeling survey," in Handbook of Software Reliability Engineering, M. R. Lyu, Ed.. New York: McGraw- Hill, 1996, pp. 71–117.

[7] Albrecht, A. J. and J. E. Gaffney. Jr. "Software Function, Source Lines of Code, And Development Effort Prediction: A Software Science Validation", IEEE Transaction on Software Engineering SE-9, 6, Nov. 1983, pp. 639-648.

[8] Halstead M., "Element of Software Science", Amsterdam: Elsevier, 1977

[9] Thomas J.McCabe, "A Complexity Measure", IEEE Transactions on Software Engineering, 1976, pages: 308-320.

[10] Gran, B. A., & Helminen, A. (2001). A Bayesian belief network for reliability assessment. Safecomp, 2187, 35–45.
[11] Zio, E. (2009). Reliability engineering: Old problems and new challenges. Reliability Engineering and System Safety, 94, 125–141.
[12] Chang-Hua Hu, Xiao-Sheng Si, Jian-Bo Yang, "System reliability prediction model based on evidential reasoning algorithm with nonlinear optimization", Expert Systems with Applications 37 (2010), pp. 2550-2562.
[13] Lata Nautiyal, Neena Gupta, "A Contemporary Certification Process for Component-Based Development", International Journal of Computer Technology & Applications, Vol 6 (1), pp. 127-132, 2015.
[14] Marko Palviainen, Antti Evesti, Eila Ovaska, "The reliability estimation, prediction and measuring of component-based software", The Journal of Systems and Software (84), pp. 1054-1070, 2011.
[15] Zhang F, Zhou X, Dong Y, Chen J. Consider of fault propagation in architecture based software reliability analysis. In: International conference computer system and application; 2009. pp.783–6.
[16] Gokhale S, Lyu M, Trivedi K. Reliability simulation of component-based software systems. In: Proceeding of ninth International Symposium on Software Reliability Engineering: ISSRE '98. Paderborn, Germany; 1998. p. 192–201.
[17] Wang D, Hang N. Reliability analysis of component based software based on rewrite logic. In: Proceeding of Twelfth IEEE International Workshop on Future Trends in Distributed Computer System; 2008. p. 126–32.
[18] Lo J-H. Early software reliability prediction based on support vector machines with genetic algorithms. In: Fifth IEEE Conference ind electron app; 2010. p. 2221–6.
[19] Goswami V, Acharya YB. Method for reliability estimation of COTS components based software systems. In: International Symposium on Software Reliability Engineering: ISSRE; 2009.
[20] Yacoub S, Cukic B, Ammar H. A scenario-based reliability analysis approach for component based-software. IEEE Transaction on Reliability 2004;53(4):465–80.
[21] Gayen T, Misra RB. Reliability assessment of elementary COTS software component. Internation Journal of Recent Trends Engineering 2009;1(2).
[22] Goseva-Popstojanova K, Trivedi K. Architecture-based approaches to software reliability prediction. IEEE Trans Software Engineering 2003;46(7):1023–36.
[23] Yacoub S, Ammar H. A methodology for architectural-level risk analysis. IEEE Trans Software Engineering 2002;28:529–47.
[24] Singh H. A Bayesian approach to reliability prediction and assessment of component based systems. In: Twelfth IEEE International Symposium on Software Reliability Engineering: ISSRE '01; 2001. p. 12–21.
[25] Everett WW. Software component reliability analysis. In: Application-specific system software engineering: ASSET '99; 1991. p. 204–11.
[26] Whittaker KR, Thomason M. A Markov chain model for predicting the reliability of multi-build software. Journal of Information Software Technology 2000;42(12): 889–894.
[27] Pressman, R. "Software Engineering: A Practitioner's Approach" McGraw Hill, Inc. Fourth Edition 1997, pp456.
[28] B. Littlewood, L. Strigini. Validation of Ultra-High Dependability for Software-based Systems. Communications of the ACM 36, 11 (November 1993), 69-80.