An Analytical Model for Estimating the Reliability of Critical Software Systems by Considering the Self-Healing Property of Bottleneck Components

Ali Tarinejad[1], Habib Izadkhah[2,*], Mohammadreza Mollahoseini Ardakani[3] , Kamal Mirzaie[4]
[1]Department of Computer Engineering, Maybod Branch, Islamic Azad University, Maybod, Iran
[2]Department of Computer Science, University of Tabriz, Tabriz, Iran
[3] Department of Computer Engineering, Maybod Branch, Islamic Azad University, Maybod, Iran
[4] Department of Computer Engineering, Maybod Branch, Islamic Azad University, Maybod, Iran
Email: ali_ahar_tarinejad@yahoo.com , izadkhah@tabrizu.ac.ir , mr.mollahoseini@iau.ac.ir , kamal.mirzaie@iau.ac.ir
Receive Date: 20 March 2022, Revise Date: 17 April 2022, Accept Date: 10 June 2022

Abstract

Architecture-based techniques for assessing the reliability of critical software systems have attracted a lot of attention in recent years due to the emerging pattern of component-based software development. In order to prevent the failure of software systems in the final phases of development in critical software systems, we must apply the software reliability evaluation process to all stages of software development. Reliability evaluation of component-based critical software systems is very important in the early stages of software system development and from its architecture as one of the quality attributes of software systems. This article proposes a method to evaluate the reliability of critical software systems by considering the self-healing effect of bottleneck components on software reliability. A self-healing component can automatically repair itself and return to a normal state when a failure occurs. Since the design of a self-healing component is very complicated and costly, it is not possible to create self-healing for all components. Therefore, identifying bottleneck components in order to self-repair them in the early stages of software development can have a great impact on reliability. Nowadays, several methods have been proposed based on design models to evaluate the reliability and software systems, but the effect of self-repair on reliability and also finding components that have a great impact on software reliability. No report has been provided for self-repairing the components in the early stages of software development. In this article, first, a method for modeling self-healing using the Markov chain will be proposed, and then four different methods (without -Taylor series - without self-healing, without Taylor series - with self-healing, with -Taylor series - without self-healing and with Taylor series - with self-healing) will be presented to evaluate the reliability of a software system based on its architecture. The relations proposed will enable a software engineer to identify the influential and bottleneck components for self-healing.

Keywords: Software reliability, Software architecture, Discrete-time Markov chain, Self-healing component, and Sensitivity analysis

1. Introduction

Due to the dependence of our daily life on the services of software systems, the reliability evaluation techniques of these software systems are of great importance. The impact of the structure of a software system on its reliability and correctness has been considered for almost two decades. The presence of software systems in equipment, devices, services and daily life activities of people has increased. Computer system failures make headlines because they cause inconvenience to people (failure of household appliances), economic damage (interruptions of banking services) and in extreme cases death (failure of flight control systems or medical software).

*- Corresponding author. *E-mail address:* izadkhah@tabrizu.ac.ir (H. Izadkhah).

It is important to evaluate the non-functional needs in the software development levels. In component-based systems, non-functional requirements such as reliability, efficiency and security determine the quality of the final product and the success of the software [1]. Component-based systems are formed from the community and the juxtaposition of existing independent components and interact with each other to provide services for users. The problem of evaluating the non-functional needs of each of the components alone is one of today's research fields, and on the other hand, even if it is assumed that a component alone has a suitable quality, the suitable quality of the combination of components with each other is not always guaranteed. Therefore, in the initial phases of development, in addition to evaluating each component, it is necessary to evaluate how they interact with each other.

Software reliability is defined as the probability that the software will perform its function correctly (without failure) during a certain period of time and under specific operational and environmental conditions that it encounters [1].

In the literature, the method of evaluating the reliability of a software system has been examined from different perspectives, which are (1) the black box perspective and (2) the white box perspective [2, 3]. Common approaches to software reliability assessment are based on the black box, that is, the software system is considered as a whole, and only its interaction with the outside world, without considering its internal structure, is modeled. is made The three main problems of these methods are that (1) because they do not know enough about the internal functioning of the software system, so they cannot be accurate enough in evaluating the reliability of a software system, (2) if after the evaluation, the engineers software come to the conclusion that the system does not have adequate reliability, replacing this software with new software or fixing the problems of the current software is not a good option because it will not be economically viable, and (3) these methods cannot be applied in the early stages of software development and from design models. In contrast to black box methods, white box methods, because they know the internal structure of a software system, can assess the reliability of the software system with acceptable accuracy. The main advantage of these methods is that they can be used in the early stages of software development. Due to the fact that checking and evaluating these features (Functional and non- Functional requirements) before the design and implementation stage, spends less time and money, the best time to assess the evaluable behavior of the system, is the time when the architecture of that software is created. Software architecture, as the first product and output of the software design stage, plays an important and direct role in the development of complex software systems, and with its help, the evaluable behavior of the system, i.e. quality attributes, can be determined, like security, reliability measures usability, changeability, and efficiency.

Most of the white box methods start to evaluate and predict the reliability of a software system based on the software architecture. Software architecture is how the main components of a software system are put together [1]. According to the IEEE standard [1], architecture means

providing a technical description of a system that shows the structure of its components, the relationship between them, and the principles and rules governing their design and evolution over time. So software architecture shows how the main components of a software system are put together. Software system clustering is the main activity of finding a suitable architecture. In fact, clustering of software classes is the process of grouping software classes so that classes with the highest degree of dependency are placed in a cluster. Clustering makes it easier to understand the software and facilitates maintenance operations in the future. This clustering is done based on the connections between classes. In general, these connections are shown in the form of Component Dependency Graphs (e.g., MDG †) where the nodes represent the software components and the edges model the connections between them [4].

1.1 Statement of the problem

Reliability evaluation from architecture in the early stages of software production plays a significant and important role in the development of software systems with high reliability [5]. Component-based development is considered the main solution to overcome major software challenges. Therefore, it is very important to provide a model for reliability evaluation for component-based software from an architectural point of view. Many methods have been presented in the literature to describe the software architecture to evaluate the reliability, which includes types of Petri nets (such as HCPN, SPN) [6], automata [7], Markov chains (such as DTMC, CTMC) [5, 8, 9], Bayesian models [10].

It is clear that the impact of components on the reliability of a software system is not the same; Also, the number of times a component is executed during the execution of a software system is different from other components. A component whose number of repetitions is high has a greater impact on the reliability of a software system. Among the mentioned models, only Markov chains have this feature, they can calculate the number of times a component is executed during the execution of a software system. Therefore, in this article, we used the discrete Markov chain for architecture modeling.

In recent years, the importance of adding self-healing capability to components has been emphasized a lot. Self-healing components try to automatically detect and repair errors that occur during their use [11-13]. In this article, we will present a method for modeling self-healing components using the Markov chain, by using which we will be able to evaluate the reliability of software systems considering the self-healing property.

Next, in Section 2, the basic concepts and previous research related to the proposed solution will be presented, in Section 3, we will present the proposed solution, in Section 4, we will present an example of how to evaluate and the practical results of the description. will be given, and at the end, in Section 5, we have given conclusions and future discussions.

□- Component dependency graphs

2. Background

In this section, the related works in the field of evaluating the reliability of software systems and also the basic concepts related to discrete-time absorbing Markov chain are examined.

2.1 Previous work

Calculating the reliability of a software system is done at two levels the component and the entire software system. At the level of the component, it is tried to predict the reliability of the component by combining the results of the analysis of the internal structure and behavior of the component with the data obtained from the test or actual historical experiences reported. In [14-18], methods to determine the reliability of a component are presented. Determining the reliability of a component is not the subject of this article. At the level of the entire software system, the goal is to find the reliability of the software system based on the configuration and interactions between the components.

In [19], existing architecture-based models are divided into three broad categories: state-based, path-based, and additive. State-based models (such as Markov chains, Petri nets, and automata) use control graphs to represent software architecture and use analytical methods to predict reliability. Path-based models calculate the reliability of the software according to the possible execution paths of the program. Execution paths may be determined using simulation, program execution, or algorithm. Incremental models assume that the reliability of each component can be modeled with a non-homogeneous Poisson process (NHPP), which causes the system failure process [20]. Among the three categories of architecture-based software reliability models, state-based models have been researched more than the other two methods.

In state-based models, the software architecture can be modeled by DTMC, CTMC, SMP, DAG, or SPN. DTMC, CTMC, and SPM can be divided into two types: irreducible and absorbent. SPN and DAG can be used to model concurrent applications. DAG is limited to modeling concurrent applications without loops, but SPN is also used for applications with loops. The failure behavior of a component can be shown by component reliability, constant failure rate and time-dependent failure severity. Reliability estimation methods in state-based approaches are divided into two categories: hybrid and hierarchical, which shows how to consider the failure behavior of components with software architecture to predict reliability. In the combined method, the failure behavior of the components is combined with the program architecture and a combined model is obtained to predict the reliability of the system. In the hierarchical method, first the software architecture is modeled by state-based models, and then the reliability is estimated from this model by considering the failure behavior of the components.

2.2 Discrete-Time Markov Chain

The Markov chain is a stochastic memory-less process. Stochastic processes refer to phenomena whose outcome is not given before they happen, such as throwing coins or dice. When the conditional probability distribution for the system state in the next step depends only on the current state of the system and is not dependent on previous states, the Markov chain is used for modeling. Discrete-time absorbent Markov chain is a good approach to describe

the structure of components of the software system and to predict its reliability description [1, 5], and it has many uses in real-world modeling. A Markov chain is a sequence of a finite or countable number of stochastic variables $\{X_n, n=0,1,2,...\}$ with Markov property as follows:

$$p\{X_{n+1}=j \mid X_n=i, X_{n-1}=i_{n-1}, \ldots, X_1=i_1, X_0=i_0\} = p\{X_{n+1}=j \mid X_n=i\} = p_{ij} \quad (1)$$

Discrete-time Markov chain is generally divided into two categories:

1) Irreducible: each mode of it can be accessed through all other modes of it.
2) Absorber: it has at least one non-transmittable mode and when reaching to this mode, its mode will not change anymore.

The conditional probability $P\{X_{n+1}=j \mid X_n=i\}$ is called one-stage transition probability, i.e. transition from state *i* in step *n* to state *j* at step *n*+1. As it is apparent from this possibility, it depends on *i*, *j*, and *n*. Matrix $P=(p_{xy})$ that its entries denotes one-stage transition probabilities is called a one-stage transition probability matrix. P is equal to:

$$P=\begin{bmatrix} p_{00} & p_{01} & p_{02} & \cdots \\ p_{10} & p_{11} & p_{12} & \cdots \\ \vdots & \vdots & \vdots & \vdots \\ p_{i0} & p_{i1} & p_{i2} & \cdots \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

If P is a transition probability matrix of a Markov chain, then we have $p^{(n)}=p^{(0)}p^{(n)}$, p^0 is the initial distribution or distribution of the initial probability and p^n is the n stepwise probability matrix of the transition.

A discrete-time absorbing Markov chain is described by its transition probability matrix. This matrix with *n* states and *m* absorbing state $(m<n)$, is divided as follows:

$$P=\begin{bmatrix} Q & C \\ 0 & 1 \end{bmatrix}$$

Where *Q* is a matrix with $(n-m)\times(n-m)$ implying the probability of transition between non-absorbing states, *C* or *R* is a matrix with $(n-m)\times m$ to show the probability of transition between non-adsorbent and absorbent states, 1 is an identity matrix or Unit matrix with, and 0 is a zero matrix with dimensions $m\times(n-m)$.

The fundamental matrix M is a matrix in which the element (i, j) represents the expected number of expected visits of j to state *i*. This matrix is defined as Eq. 2:

$$M=(I-Q)^{-1}=I+Q+Q^2+\ldots=\sum_{k=0}^{\infty}Q^k \quad (2)$$

The variance of expected expectations from the matrix M is defined in accordance with Eq. 3:

$$\sigma^2=M(2M_{dg}-I)-M_{sq} \quad (3)$$

where M_{dg} implies a diagonal matrix and M_{sq} represents a square matrix.

Suppose $X_{i,j}$ represents expect the number of jumps (the mathematical hope) of state j starting from the state i before entering an absorbing state, which is determined using the element (i, j) of the matrix M.

$$E[X_{i,j}]=m_{i,j} \quad (4)$$

3. Proposed Approach

The general aims to calculate the reliability of a software system by considering the self-healing property of the most influential components. The flowchart of the proposed approach is shown in Figure 1. Details of each of the parts of this flowchart are described in the following.

Step 1: First, we extract the software architecture using the Bunch tool [1], and convert this architecture to a discrete time-absorbing Markov chain and calculate the transition probability matrix, fundamental matrix, and variance matrix. Creating a discrete Markov chain from software architecture is as follows:

1) Each component in architecture will be equivalent to each state in the Markov chain,
2) Let Fan-in and Fan-out represent the number of calls between the two components x and y and the number of outgoing calls from component x, respectively, in architecture. The probability of transfer between x and y is determined by Eq. 5.

[Fan-in / Fan-out] (5)

For a software system including a number of clusters, we can show the structure of the software system (software architecture) with a Markov chain. The states of the Markov chain represent the clusters and the edges between the states represent the transfer of control from one cluster to another. For example, consider the famous traveling salesman problem (TSP). First, we extract the TSP structure from its source code using the Bunch tool. The input of the Bunch tool is the call graph and its output is the software structure (software architecture). The commercial tool NDepend [21] was used to extract the TSP call graph from its source code. The NDepend tool for most of the world's famous programming languages can extract the call graph from the source code. After extracting the call graph, it should be clustered to extract the appropriate architecture. Figure 1 shows the extracted architecture for the TSP problem from its source code.

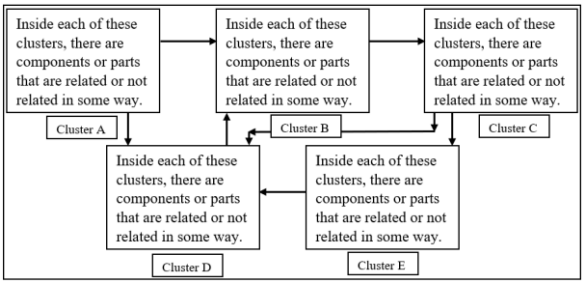


Fig.1. Architecture of the software system for the Travelling salesman problem using Bunch

After extracting the architecture, we convert them into Markov chains. The Markov chain for figure 1 is as figure 2.

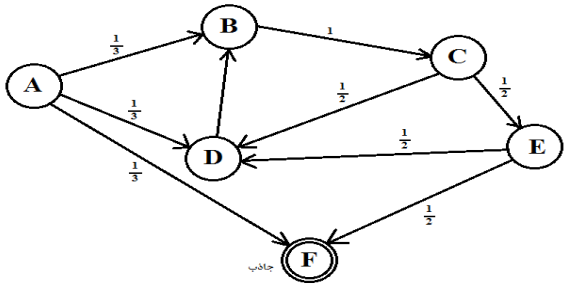
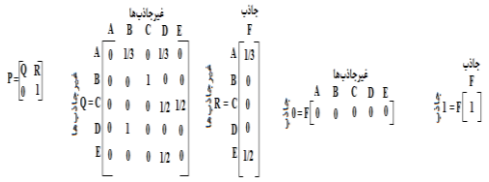


Fig.2. Markov chain Figure 1

The numbers on the edges indicate the probability of moving from one cluster to another. For example, in Figure 1, a total of 3 edges were removed from cluster 3, 1 of which went to cluster 4, 1 to cluster 5, and 1 edge to cluster 6. Cluster 6 is added as a final state to the Markov chain, which is not usually represented in architecture. Each of A, B, C, D, and E are clusters that have at least one or more components inside them, cluster F is the final state in the Markov chain, which has no output edge and is absorbed. We find the transfer probability matrix P in Figure 2 as follows:



The first solution to find the basic matrix M by the inverse method: now we insert the matrices Q, R, 1, and 0 in the matrix P and we have:

$$P=\begin{bmatrix} Q & R \\ 0 & 1 \end{bmatrix}=\left[\begin{array}{ccccc|c} 0 & 1/3 & 0 & 1/3 & 0 & 1/3 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 1/2 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 0 & 1/2 \\ \hline 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right]$$

Note: In the P matrix, the sum of the row probabilities must be equal to 1. And the probability of each element must be greater than 0 and less than 1.

Now, if we want to calculate the basic matrix M by the inverse matrix method, that is, M = (I - Q)-1,

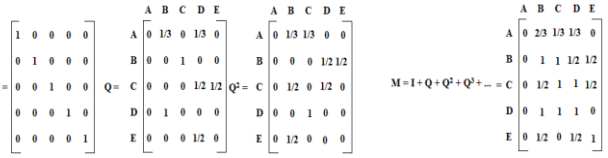
$$I=\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad Q=\begin{bmatrix} 0 & 1/3 & 0 & 1/3 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 1/2 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 0 \end{bmatrix} \quad M=(I-Q)^{-1}=\begin{bmatrix} 1 & -1/3 & 0 & -1/3 & 0 \\ 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1/2 & -1/2 \\ 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1/2 & 1 \end{bmatrix}$$

Solving this inverse matrix manually is very time-consuming, one way to solve this is to use MATLAB software as follows:

```
>> I=eye(5)

I =

     1     0     0     0     0
     0     1     0     0     0
     0     0     1     0     0
     0     0     0     1     0
     0     0     0     0     1

>> Q=[0 1/3 0 1/3 0;
0 0 1 0 0;
0 0 0 1/2 1/2;
0 1 0 0 0;
0 0 0 1/2 0]

Q =

         0    0.3333         0    0.3333         0
         0         0    1.0000         0         0
         0         0         0    0.5000    0.5000
         0    1.0000         0         0         0
         0         0         0    0.5000         0

>> M=inv(I-Q)

M =

    1.0000    2.6667    2.6667    2.3333    1.3333
         0    4.0000    4.0000    3.0000    2.0000
         0    3.0000    4.0000    3.0000    2.0000
         0    4.0000    4.0000    4.0000    2.0000
         0    2.0000    2.0000    2.0000    2.0000
```

Another method is to use the series expansion or the same Eq. 4, which is enough for about 3 sentences.

The second solution to find the basic matrix M by Eq. 4: because our Q matrix is a 5x5 matrix, then our corresponding matrix I also becomes a 5x5 matrix, and we have:



And if we calculate the calculations of each element, the matrix M will be as follows:

```
>> M=[1 2/3 1/3 1/3 0;
0 1 1 1/2 1/2;
0 1/2 1 1 1/2;
0 1 1 1 0;
0 1/2 0 1/2 1]

M =

    1.0000    0.6667    0.3333    0.3333         0
         0    1.0000    1.0000    0.5000    0.5000
         0    0.5000    1.0000    1.0000    0.5000
         0    1.0000    1.0000    1.0000         0
         0    0.5000         0    0.5000    1.0000
```

The variance of the number of expected meetings of the (i, j)th element is calculated from the matrix M as follows, where M_{dg} is the diagonal matrix of M and M_{sq} is the square (quadratic) of the basic matrix of M.

```
M =

    1.0000    2.6667    2.6667    2.3333    1.3333
         0    4.0000    4.0000    3.0000    2.0000
         0    3.0000    4.0000    3.0000    2.0000
         0    4.0000    4.0000    4.0000    2.0000
         0    2.0000    2.0000    2.0000    2.0000

>> Mdg=[1 0 0 0 0;
0 4 0 0 0;
0 0 4 0 0;
0 0 0 4 0;
0 0 0 0 2]

Mdg =

     1     0     0     0     0
     0     4     0     0     0
     0     0     4     0     0
     0     0     0     4     0
     0     0     0     0     2

>> variance=M*(2*Mdg-I)-(M^2)

variance =

         A         B         C         D         E
    A    0  -14.6667  -17.3333  -14.0000  -15.3333
    B    0  -16.0000  -20.0000  -19.0000  -20.0000
    C    0  -19.0000  -16.0000  -16.0000  -18.0000
    D    0  -20.0000  -24.0000  -16.0000  -22.0000
    E    0  -12.0000  -14.0000  -10.0000  -10.0000
```

Table 1: Probabilities of transmission between components for pacemaker software [7]

$p_{Start,Prog}=0.01$	$p_{Start,AR}=0.64$	$p_{Start,VT}=0.35$	$p_{Prog,RS}=0.002$	$p_{Prog,CD}=0.008$
$p_{Prog,T}=0.99$	$p_{AR,VT}=0.19$	$p_{AR,Heart}=0.47$	$p_{AR,T}=0.34$	$p_{VT,AR}=0.29$
$p_{VT,Heart}=0.29$	$p_{VT,T}=0.42$	$p_{RS,CD}=0.005$	$p_{RS,CG}=0.005$	$p_{CG,VT}=0.0025$
$p_{CD,Prog}=0.008$	$p_{CD,CG}=0.002$	$p_{CD,T}=0.99$	$p_{CG,AR}=0.0025$	$p_{Heart,VT}=0.64$
$p_{CG,RS}=0.005$	$p_{CG,CD}=0.005$	$p_{CG,T}=0.99$	$p_{Heart,AR}=0.35$	$p_{T,T}=1.00$
$p_{Heart,T}=0.01$				

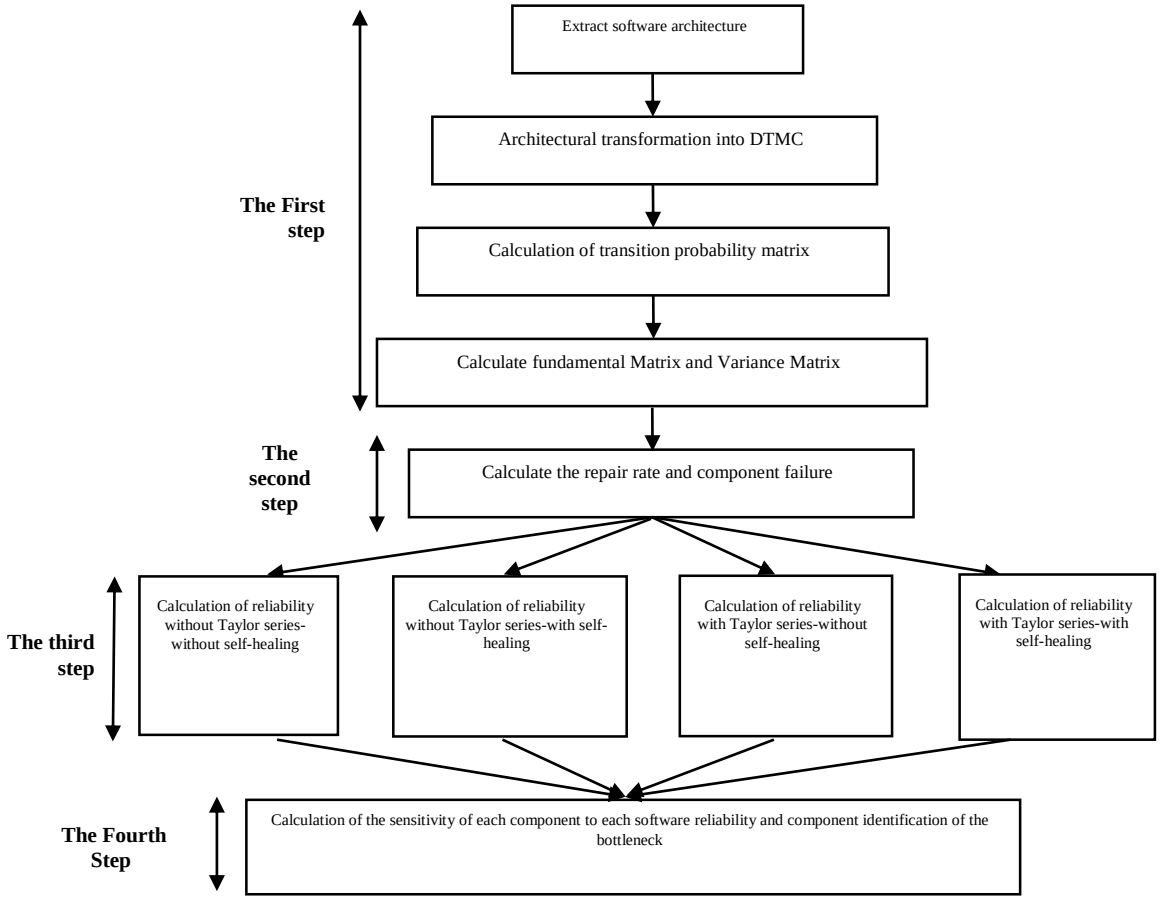


Fig.3. The proposed process steps

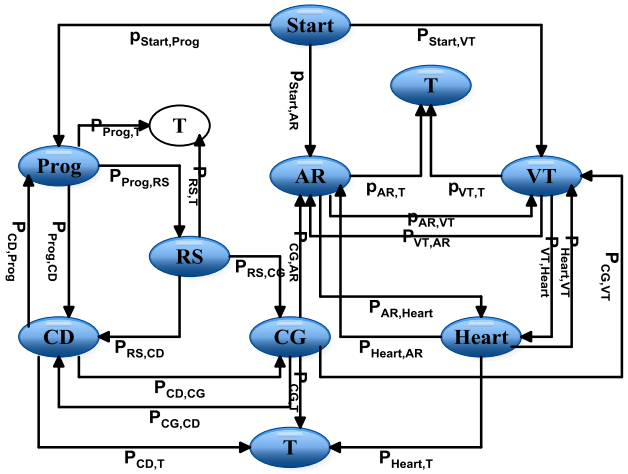


Fig4.Equivalent Markov chain for pacemaker software architecture

Case study. The pacemaker is a device for placement in the heart that helps the cardiac function when the heart's natural rate is low due to the disease. The pacemaker is consisting of the following components: reed switches (RS) coil driver (CD), communication gnome (CG), ventricular model (VT), and atrial model (AR) [7]. Figure 4 shows the Markov chain for the pacemaker application. The architecture of this artificial pacemaker has nine components that in an equivalent Markov chain has turned into nine modes. The inter-components transition probabilities for pacemaker architecture are shown in Table 1. These possibilities are obtained from data collected during the simulation of the characteristics of an artificial pacemaker. The reliability of each component and other parameters are shown in Table 2. Where $m_{1,i}$ ($x_{1,i}$) is equal to the expected number of encounters from each component starting from the first component, is equal to the root of (1, i) or the first row of the basic matrix M and $Var_{1,i}^2$ is the first row of the variance matrix.

Table 1: R_i, $X_{1,i}$, $Var_{1,i}^2$, m_i, n_i initial and m_i, n_i new

Comp	R_i	$X_{1,i}$	$\sigma_{1,i}^2$	m_i initial	n_i initial	m_i new	n_i new
Start	1.0000	1.0000	0.0000	0.9750	0.0131	0.9867	0.0133
Prog	0.9000	0.0100	0.0099	0.8503	0.0231	0.9736	0.0264
AR	0.9000	1.3481	1.3315	0.8999	0.0345	0.9631	0.0369
VT	0.9000	1.2422	1.4658	0.8806	0.0254	0.9720	0.0280
RS	0.9000	0.0000	0.0000	0.9003	0.0354	0.9622	0.0378
CD	0.9000	0.0001	0.0001	0.8904	0.0209	0.9771	0.0229
CG	0.9000	0.0000	0.0000	0.8807	0.0887	0.9085	0.0915
Heart	1.0000	0.9938	1.8649	0.9508	0.0194	0.9800	0.0200
T	1.0000	1.0000	0.0000	0.9678	0.0208	0.9790	0.0210

Step 2: Repair Rate and Failure Rate: in this step, we propose a method to calculate the self-healing effect on the components. A component-based software system consisting of a combination of several components and each component has a failure rate and repair rate. Since component reliability improves with self-healing properties, it is, therefore, necessary to calculate the reliability of the self-heal component. Let m_i and n_i denote the failure rate and the repair rate of each component, respectively. In the beginning, components are healthy and perform their tasks properly, due to programming errors or other reasons, the software system can fail. Depending on its functionality, it is likely to fail if one or more components of the system fail. So, for each component, the repair rate is considered to be the probability that the component will return to the correct state if a breakdown occurs (i.e., the component can repair itself and continue working). In other words, when the component is in a failure state, the component is likely to repair and return to its safe state.

In a software system, the component mode in the provision of services depends on the current state, the failure rate, and the repair rate of the component, and does not depend on the time and the component states in previous references, therefore is a stochastic process. It can be modeled with a discrete-time Markov chain. In this way, X_n is the component state in the reference n which takes its values from {0, 1}.

$$\{X_n: \forall\, n>=0\},\ M=\{0,1\} \qquad (6)$$

About solving the reliability of the self-healing component, we must know that starting from the safe state of a component, what is the probability of the component being referenced in a safe state. So, we have:

$$P(X_{n+1}=j \mid X_n=i)=p_{ij} \qquad (7)$$

We consider two states, namely state 1 and state 0, for self-healing components, which show, respectively, safe and failure states. The single-step probability matrix for these two states can be as follows.

$$P = \begin{bmatrix} p_{00} & p_{01} \\ p_{10} & p_{11} \end{bmatrix} \qquad P = \begin{bmatrix} 1-m_i & m_i \\ n_i & 1-n_i \end{bmatrix}$$

The Markov chain for matrix P is shown in Figure 5.

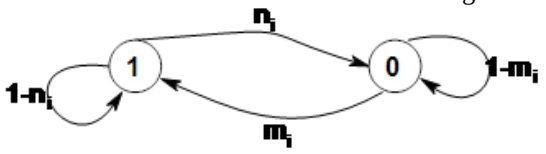


Fig.5. One-step probability matrix Markov chain for self-healing components

Calculating repair rate and failure rate: according to a proposition in the Markov chain model, for an irreducible Markov chain with a finite number of states, the stable state is unique.

$$\prod_j = \sum_{i=1}^{n} \prod_i P_{i,j} \quad ,j \in M, \qquad (8)$$
$$\sum_{i=1}^{n} \prod_j = 1$$

According to the transition matrix and using equations relating to the distribution of stable state, this value with respect to equations of the Markov model for the component can be computed as follows:

$$\begin{aligned} &(1-n_i)\Pi_1+m_i\,\Pi_0 = \Pi_1 \\ &n_i\,\Pi_1+(1-m_i)\,\Pi_0 = \Pi_0 \\ &\Pi_0+\Pi_1 = 1 \end{aligned} \qquad (9)$$

and as a result

$$\Pi = [\Pi_0\ ,\ \Pi_1]=[\ \frac{n_i}{m_i+n_i}, \frac{m_i}{m_i+n_i}] \qquad (10)$$

and value $\Pi 1 = \frac{m_i}{m_i+n_i}\,\pi 1$ is called probability in the safe state.

Step 3: In this step, four metrics considering different combinations of Taylor series and self-healing are proposed to evaluate reliability of a software system from its architecture. Let Ri denotes the reliability of component i in the software system. Then, the overall reliability of this system is calculated as follows:

$$R = \prod_{i=1}^{n} R_i \tag{11}$$

Because components with a high number of repetitions during a typical run have a great impact on the reliability of a software system, to improve accuracy, we consider the number of repetitions of each component. Let m1,i denotes the expected number of visits of component i starting from the first component. This value is equal to entry (1, i) from the fundamental Matrix M. Thus, the overall reliability of a software system will be as follows:

$$R = \prod_{i=1}^{n} \left[{R_i}^{m_{1,i}} \right] \tag{12}$$

Given that we use the static structure of software to predict its reliability and it is not possible to accurately determine the reliability of each component and the number of repetitions of each component in the design stage; so, to reduce error in the estimation of reliability, we use second-order and third-order Taylor series approximation. Let Ri, Var(1, i), m_i and n_i, respectively, indicate the reliability of component i, expected visit variance of component i, the healing rate of component i, and the failure rate of component i.

Equations 13 to 18 represent metrics for calculating reliability with consideration of the following combined modes: without Taylor series- without self-healing, with second-order Taylor series-without self-healing, with third-order Taylor series-without self-healing, with Taylor series-with self-healing, with second-order Taylor series-with self-healing, with third-order Taylor series-with self-healing.

I. without Taylor series- without self-healing

$$R = \prod_{i=1}^{n} \left[{R_i}^{m_{1,i}} \right] \tag{13}$$

II. with second-order Taylor series-without self-healing

$$R = \prod_{i=1}^{n} \left[{R_i}^{m_{1,i}} + \frac{1}{2}\left({R_i}^{m_{1,i}} \right).(\log R_i)^2 .\sigma_{1,i}^2 \right] \tag{14}$$

III. with third-order Taylor series-without self-healing

$$R = \prod_{i=1}^{n} \left[\begin{array}{l} {R_i}^{m_{1,i}} + \frac{1}{2}\left({R_i}^{m_{1,i}} \right).(\log R_i)^2 .\sigma_{1,i}^2 + \frac{1}{6}\left(\sigma_{1,i}^2 \right). \\ \left[\left(m_{1,i} .{R_i}^{(m_{1,i})-1} \right).(\log R_i)^2 + \left(2\frac{\log R_i}{R_i} \right).\left({R_i}^{m_{1,i}} \right) \right] \end{array} \right] \tag{15}$$

By replacing the self-healing relationship obtained, $R_i = \frac{m_i}{m_i + n_i}$, into Eq. 13-15, respectively, Eq. 16-18 are obtained.

IV. without Taylor series-with self-healing

$$R = \prod_{i=1}^{n} \left(\frac{m_i}{m_i + n_i} \right)^{m_{1,i}} \tag{16}$$

V. with second-order Taylor series-with self-healing

$$R = \prod_{i=1}^{n} \left[\left(\frac{m_i}{m_i + n_i} \right)^{m_{1,i}} + \frac{1}{2}\left(\left(\frac{m_i}{m_i + n_i} \right)^{m_{1,i}} \right).\left(\log\left(\frac{m_i}{m_i + n_i} \right) \right)^2 .\sigma_{1,i}^2 \right] \tag{17}$$

VI. with third-order Taylor series-with self-healing

$$R = \prod_{i=1}^{n} \left[\begin{array}{l} \left(\frac{m_i}{m_i + n_i} \right)^{m_{1,i}} + \frac{1}{2}\left(\left(\frac{m_i}{m_i + n_i} \right)^{m_{1,i}} \right).\left(\log\left(\frac{m_i}{m_i + n_i} \right) \right)^2 .\sigma_{1,i}^2 + \frac{1}{6}\left(\sigma_{1,i}^2 \right). \\ \left[\left(m_{1,i} .\left(\frac{m_i}{m_i + n_i} \right)^{(m_{1,i})-1} \right).\left(\log\left(\frac{m_i}{m_i + n_i} \right) \right)^2 + \left(2\frac{\log\left(\frac{m_i}{m_i + n_i} \right)}{\left(\frac{m_i}{m_i + n_i} \right)} \right).\left(\left(\frac{m_i}{m_i + n_i} \right)^{m_{1,i}} \right) \right] \end{array} \right] \tag{18}$$

Step 4- Sensitivity analysis: based on metrics presented in Step 3, in this step to calculate the impact of reliability of each component on the reliability of the entire software system, four metrics are presented. The sensitivity analysis is used to identify the bottleneck components.

Generally, the effect of a change in the reliability of component k, Rk, on the expected reliability of the software can be stated by the differential of software reliability as follows:

$$\frac{dE[R]}{dR_k} \qquad (19)$$

By calculating differential Rk relative to the entire of the software system, the following metrics are obtained:

A. sensitive impact on the reliability of a component, without Taylor series-without self-healing

$$\frac{dE[R]}{dR_k} = \left[m_{1,k} \cdot R_k^{(m_{1,k})-1} \right] \times \left[\prod_{i=1, i \neq k}^{n} R_i^{(m_{1,i})} \right] \qquad (20)$$

B. sensitive impact on the reliability of a component, with second-order Taylor series-without self-healing

$$\frac{dE[R]}{dR_k} = \left[m_{1,k} \cdot R_k^{(m_{1,k})-1} + \frac{1}{2}\sigma_{1,k}^2 \cdot \left(m_{1,k} \cdot R_k^{(m_{1,k})-1} \cdot (\mathrm{Log} R_k)^2 + \frac{2 Log R_k}{R_k} \cdot R_k^{(m_{1,k})} \right) \right] \times \left[\prod_{i=1, i \neq k}^{n} \left(R_i^{(m_{1,i})-1} + \frac{1}{2} R_i^{(m_{1,i})} \cdot (\mathrm{Log} R_i)^2 \cdot \sigma_{1,i}^2 \right) \right] \qquad (21)$$

C. sensitive impact on the reliability of a component, without Taylor series-with self-healing

$$\frac{dE[R]}{dR_k} = \left[m_{1,k} \cdot \left(\frac{m_k}{m_k + n_k} \right)^{(m_{1,k})-1} \right] \times \left[\prod_{i=1, i \neq k}^{n} \left(\frac{m_i}{m_i + n_i} \right)^{(m_{1,i})} \right] \qquad (22)$$

D. sensitive impact on the reliability of a component with second-order Taylor series-with self-healing,

$$\frac{dE[R]}{dR_k} = \left[m_{1,k} \cdot \left(\frac{m_k}{m_k + n_k} \right)^{(m_{1,k})-1} + \frac{1}{2}\sigma_{1,k}^2 \cdot \left(m_{1,k} \cdot \left(\frac{m_k}{m_k + n_k} \right)^{(m_{1,k})-1} \cdot \left(\mathrm{Log} \frac{m_k}{m_k + n_k} \right)^2 + \frac{2 Log \frac{m_k}{m_k + n_k}}{\frac{m_k}{m_k + n_k}} \cdot \left(\frac{m_k}{m_k + n_k} \right)^{(m_{1,k})} \right) \right] \times \left[\prod_{i=1, i \neq k}^{n} \left(\left(\frac{m_i}{m_i + n_i} \right)^{(m_{1,i})-1} + \frac{1}{2} \left(\frac{m_i}{m_i + n_i} \right)^{(m_{1,i})} \cdot \left(\mathrm{Log} \frac{m_i}{m_i + n_i} \right)^2 \cdot \sigma_{1,i}^2 \right) \right] \qquad (23)$$

4. Evaluation and practical results

In this section, we examine the three relationships of the third step to calculate software reliability using a case study, and by applying the data of Tables 1 and 2 to Eq. 13 to 18, the reliability values in shown in Figure 6, and at the same time, the sensitivity analysis, i.e. the effect of the components affecting the reliability of this study according to the Eq. 20 to 23, is shown in Figures 7 and 8. Suppose pi is the number of failures observed in component i. and qi represent the estimated number of visitors to component i. This value can be calculated by Markov analysis (DTMC absorption). The reliability of each component is valued using Eq. 24 [22].

$$R_i = 1 - \lim_{n_{i \to \infty}} \frac{q_i}{p_i} \qquad (24)$$

1.4 Evaluation

The number of components of a software system cannot be large. For example, in a university system, the components can be educational assistant, research assistant, financial assistant, graduation affairs, nutrition, library, etc., which are related to each other.

Case study: A pacemaker is an implanted device that assists cardiac functions when the underlying pathologies make the intrinsic heartbeats low. Figure 5 shows the pacemaker architecture. The pacemaker consists of the following components:

• Reed Switch (RS): A magnetically activated switch that must be closed before programming the device. The switch is used to avoid accidental programming by electric noise.

• Coil Driver (CD): Receives/sends pulses from/to the device programmer. These pulses are counted and then interpreted as a bit of value zero or one. These bits are then grouped into bytes and sent to the communication gnome. Positive and negative acknowledgements,

as well as programming bits, are sent back to the programmer to confirm whether the device has been correctly programmed and the commands are validated.

• Communication Gnome (CG): Receives bytes from the coil driver, verifies these bytes as commands, and sends the commands to the Ventricular and Atrial models. It sends positive and negative acknowledgments to the coil driver to verify command processing.

• Venrticular Model (VT) and Atrial Model (AR): These two components are similar in operation. They both could pace the heart and/or sense heartbeats. Once the pacemaker is programmed, the magnet is removed from the Reed Switch. The Atrial Model and Ventricular Model communicate together without further intervention. Only battery decay or some medical maintenance reasons force reprogramming.

In addition to the above components, a dummy start component is added to model the three modes of operation of the pacemaker. These modes include the programming mode or one of the operational modes. During programming, the programmer specifies the type of the operation mode in which the device will work. The operation mode depends on whether the Atrium (A), Ventricle (V), or both are being monitored or paced. The programmer also specifies whether the pacing is inhibited (I), triggered (T), or dual (D). For example, in the AVI operation mode, the Atrial portion (A) of the heart is paced (shocked), the Ventricular portion (V) of the heart is sensed (monitored), and the Atrial is only paced when a Ventricular sense does not occur (inhibited mode). The architecture also includes the heart as an external component to/from which pulses are sent/received. A dummy terminator state is also added to indicate the termination of the pacemaker operation.

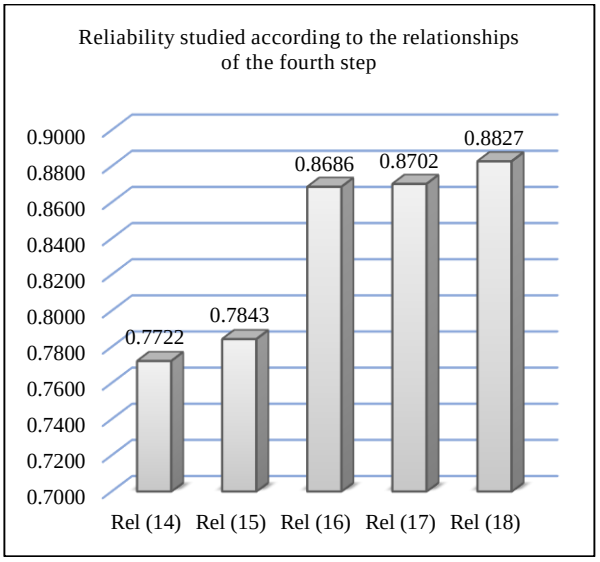


Fig.6.Reliability diagram of the Case study software system

2.4 Discussion

According to the results, the self-healing property plays an important role in the reliability of a software system, and also by considering the second and third order architecture (second and third order Taylor series), a more accurate estimate of reliability is obtained. According to the relationships obtained from the 3rd step of the fourth step, we obtain the effect value of the reliability of each component in relation to the reliability of the entire software system in order to identify the bottleneck components (Figures 7 and 8). From these figures, it is clear that in the case of the study, AR and VT components have a great impact on the reliability of the software system. In fact, reducing the reliability of these components has a great impact on reducing the reliability of the entire software system, and it is suggested that these components are self-repaired.

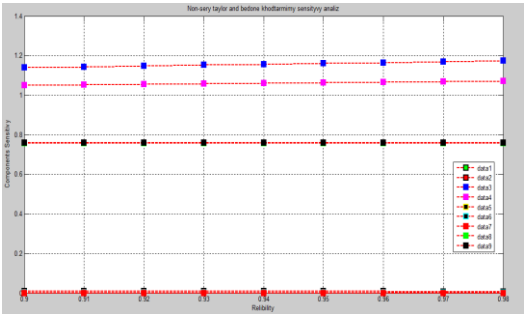


Fig.7.Sensitivity, without Taylor series - without self-healing (study case)

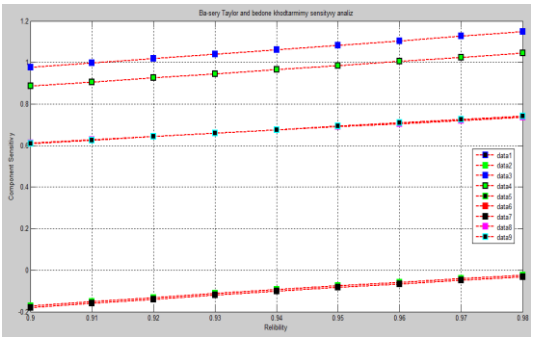


Fig.8. Sensitivity, with Taylor series - without self-healing (study case)

Using Taylor's series in calculating the reliability of a software system can increase the accuracy of reliability assessment. It considers more parameters to calculate the reliability (such as the number of runs of a component and the variance of the number of runs). Figure 6 shows that using a higher order of Taylor series improves the evaluation accuracy. Of course, because the third-order Taylor series requires higher-order derivatives, its calculation is more complicated than the second-order Taylor series. Equations 14 to 18 in Figure 6 show the effect of self-healing on reliability calculation. From these figures, it is clear that adding self-healing properties to components increases the reliability of a software system significantly. By adding the self-healing capability to the case study, for example, according to the third-order Taylor series (i.e., Eq. 18, the reliability is improved.

5 Conclusions and Future Works

In this article, the structure of the system, the time spent in each component per visit, the reliability and the failure rate of each component are determined in such a way that the resulting architecture is converted into a time-absorbing discrete Markov chain. Effective relationships were proposed to calculate the reliability of a software system in different states. Also, relationships were proposed to find bottleneck components. The development of techniques to estimate the reliability of software systems with multimodal architecture and the description of the software reliability assuming random instead of deterministic components are topics for future research.

Reference

[1] Isazadeh A, Izadkhah H, Elgedawy I., "Source Code Modularization Theory and Techniques", Springer, ISBN 978-3-319-63344-2., 2017.

[2] Patel, D., "Software Reliability: Models", International Journal of Computer Applications, 152(9), 2016.

[3] Robidoux, R., Xu, H., Xing, L. and Zhou, M., "Automated modeling of dynamic reliability block diagrams using colored Petri nets", IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans, Vol. 40, No. 2, pp.337-351, 2010.

[4] Mancoridis, Spiros, Brian S. Mitchell, Chris Rorres, Y. Chen, and Emden R. Gansner. "Using automatic clustering to produce high-level system organizations of source code." In International Conference on Program Comprehension, pp. 45-45. IEEE Computer Society, 1998.

[5] Vibhu Saujanya Sharma, Kishor S. Trivedi., "Quantifying software performance, reliability and security:An architecture-based approach", The Journal of Systems and Software, Vol. 80, pp. 493–509, 2007.

[6] Trivedi, Kishor S., and Andrea Bobbio. "DSN 2016 Tutorial: Reliability and Availability Modeling in Practice", Dependable Systems and Networks

Workshop, 2016 46th Annual IEEE/IFIP International Conference on. IEEE, 2016.

[7] Wason, Ritika, P. Ahmed, and M. Qasim Rafiq. "Automata-based software reliability model: the key to reliable software", International Journal of Software Engineering and Its Applications 7.6, pp. 111-126, 2013.

[8] SWAPNA S. GOKHALE., "Software Reliability Analysis Incorporating Second–Order Architectural Statistics", International Journal of Reliability, Quality and Safety Engineering @ World Scientific Publishing Company, USA 2010.

[9] Yakovyna, V., Fedasyuk, D., Nytrebych, O., Parfenyuk, I. and Matselyukh, V., "Software reliability assessment using high-order Markov chains", International Journal of Engineering Science Invention, 3(7), pp.1-6, 2014.

[10] Sander, P. and Badoux, R. eds., " Bayesian methods in reliability", Springer Science & Business Media, Vol. 1, 2012.

[11] Monperrus, M., "Automatic software repair: a bibliography", ACM Computing Surveys, 2017.

[12] Al-Jumeily, D., Hussain, A. and Fergus, P., "Using adaptive neural networks to provide self-healing autonomic software". International Journal of Space-Based and Situated Computing, 5(3), pp.129-140, 2015.

[13] Drew, P., Walker, T. and Ogden, R., "Self-repair and action construction", Conversational repair and human understanding, pp.71-94, 2013.

[14] Zhou, K., Wang, X., Hou, G., Wang, J. and Ai, S., "Software Reliability Test Based on Markov Usage Model. " JSW, 7(9), pp.2061-2068, 2012.

[15] Seth, K., Sharma, A. and Seth, A., "Minimum Spanning Tree-Based Approach for Reliability Estimation of COTS-Based Software Applications". IUP Journal of Computer Sciences, 4(4), 2010.

[16] Antony, J.P., "Predicting reliability of software using thresholds of CK metrics". International Journal of Advanced Networking and Applications, 4(6), p.1778, 2013.

[17] Antony, J., and H. Dev., "Estimating reliability of software system using object oriented metrics", Int. J. Comput. Sci. Eng, 283-294, 2013.

[18] Johny Antony P, Harsh Dev. "Estimating Reliability of Software System using object-oriented metrics", International Journal of Computer Science Engineering and Information Technology Research. Vol. 3, No 2, pp. 283-294, 2013.

[19] Swapna S. Gokhale and Kishor S. Trivedi., "Analytical Models for Architecture-Based Software Reliability Prediction: A Unification Framework", IEEE TRANSACTIONS ON RELIABILITY, VOL. 55, NO. 4, pp. 578-590, USA 2009.

[20] Ravishanker, N., Liu, Z. and Ray, B.K., "NHPP models with Markov switching for software reliability", Computational Statistics & Data Analysis, 52(8), pp.3988-3999, 2008.

[21] Johny Antony P, Harsh Dev. "Estimating Reliability of Software System using object-oriented metrics. " International Journal of Computer Science Engineering and Information Technology Research. Vol. 3, No 2, pp. 283-294, 2013.

[22] Yacoub S, Cukic B, Ammar H. "A scenario-based reliability analysis approach for component-based software." IEEE Trans Reliability 2004;53(4):465–80.

Ali Tarinezhad is Ph.D. student at the Department of Computer Engineering, Islamic Azad University, Maybod Branch, Iran. His-main interests and activities are in the field of advanced software engineering, reliability, and security.

Habib Izadkhah is Assistant Professor at the Department of Computer Science, University of Tabriz, Iran. His-research interests include software engineering, software architecture, reverse engineering, and organic software. More recently he has been working on the application of reverse engineering and modularization to the extract of software structure for large scale and multi programming languages source code.

Mohammadreza Mollahoseini Ardakani is Assistant Professor at the Department of Computer Engineering, Islamic Azad University, Maybod Branch, Maybod, IRAN. His-research interests are Software Engineering, Cloud Computing, Fog/Edge Computing, and Service Oriented Architecture.

Kamal Mirzaie is Assistant Professor at the Department of Computer Engineering, Maybod Branch, Islamic Azad University, Maybod, Iran. His-research interests include cognitive science, soft computing, medical data mining, parallel processing, image processing, and pattern recognition.