Reliability Evaluation of Software Architectural Styles Based on Correlated Component Failure

Sima Emadi*
Department of Computer Engineering, Yazd Branch, Islamic Azad University, Yazd, Iran.
emadi@iauyazd.ac.ir
(035-31872707, 09133532256)

Abstract this study aims to provide an efficient and scalable way to evaluate the reliability of different software architectural styles concerning correlated components failures. In this way, a method based on the discrete-time Markov chain (DTMC) model is proposed. In the proposed method, software architecture styles are used for reliability evaluation. The four main styles are transformed into Markov chain models and the transfer matrix is created for them, then using the Bernoulli distribution, the correlation between components is shown in the matrix and used in the evaluation process. The proposed method is scalable such that it can be used for large software architectures with heterogeneous and homogeneous styles. The results of the evaluation on the case study show that this method is more accurate than the other methods for the reliability prediction of the software architectures. As a result, it is concluded that the proposed method is suitable for the preliminary estimation of the software architecture reliability and can make a better comparison between various architectural styles to choose the most suitable one from the available options.

Keywords: Software architecture styles, Reliability evaluation, Correlated component failures, Discrete-Time Markov Chain, Quality Attributes, Software Architecture Design.

1. Introduction

Analysis, design, and implementation of software systems take place to solve various problems and data processing. Over time, the number of elements and components of software systems has increased and the structure of these elements and organizing software has become more complex. In order to control the complexity of software, software architecture is needed [1]. Software architecture in the software development life cycle, by determining the priority requirements, refers to the fundamental structures of a software system and the discipline of creating such structures and systems. This structure, which shows the functional and non-functional requirements of the software comprises components, relations among them, and properties of both elements and relations. Also, it provides an abstraction to manage the system complexity and establish a communication and coordination mechanism among components [2]. Since software architecture has an important role in satisfying the quality attributes of a software system (i.e. reliability, security, performance, availability, usability, and maintainability), its analysis and evaluation is necessary to determine the level of satisfaction of these quality attributes based on the structure and correlation between its components, because changing a design flaw, especially in large-scale software systems, in the implementation and testing phase is more expensive to make changes than at the architectural design stage [3].

However, designing software architecture is a complex activity during software development. Because in the process of determining the components and the relationships between them, the non-functional requirements (quality attributes) of the software must also be considered and software architectures must create based on quality attributes and impact them.

Architectural styles and patterns provide a way for how to organize software components so that the designer can easily build the software and satisfy customer functional and non-functional requirements. Software architecture styles at the highest level of granularity, show the layers and high-level modules of software systems, relationships between them, and interaction with each other.

Clement [4] defined the software architectural style in this way: “A set of constraints you put on your development to elicit desirable properties from your software architecture. These constraints may be topological, behavioral and
communication-oriented”. Also, “A style is a specialization of element and relationship types, together with constraints on how they may be used. By identifying element and relationship types, styles identify the architectural structures that architects design to achieve the system’s quality and behavioral goals.”

Moreover, styles address non-functional requirements (i.e. reliability, security, performance, availability, usability, and maintainability) and narrow the solution space when creating architecture, because architectural styles define the elements in architecture and the rules of how they interact [5]. Therefore, choosing an appropriate architectural style is one of the important decisions in software architecture design [6]. In addition to the description of the software and decomposing it into components, software architectural styles have a major impact on qualitative attributes of designed software (e.g., Reliability, performance, security) [3]. Because architectural styles in the software architecture design process guarantee the satisfaction of their quality requirements. The use of a method or a model for the evaluation of qualitative attributes in styles enables software designers to make more accurate design decisions and select the appropriate style for implementation according to the desirable qualitative attributes of the system [7].

Given the multiplicity of qualitative attributes, reliability as one of the effective attributes in determining the quality of the software is defined as “the probability of the failure-free software operation in a specified environment and at a period of time” [8][9]. Empirical studies indicate that because the components of a software system are correlated, component failures are often correlated [10]. On the other hand, a majority of the software reliability assessment approaches assume that component failures are independent, while recent studies indicate that component failures are often correlated with other components [10]. Since the correlated component failures (COCOF) may be effective in estimating the reliability of the software, they should be explicitly incorporated in the reliability evaluation [11]. Fiondella et al. [10] claim that most contemporary analysis approaches are not scalable because they require a large number of parameters and are computationally inefficient, and these approaches require an explicit characterization of the joint distribution of system components. Joint distribution is the probability of combining all failure states of software components, which is characterized by using two methods. In [12], the probability of joint distribution is calculated based on the reliability knowledge of individual components and the communication between them. In [13], the probability of joint distribution is estimated directly instead of computing component parameters. Each of these methods requires exponential computing for software consisting of \( n \) components. The drawbacks of methods show that a technique that requires less parameter and has efficient computation is necessary to analyze the reliability of software based on a large-scale that also considers the correlated component failures. Therefore, they present an efficient methodology based on the multivariate Bernoulli (MVB) distribution [14] to analyze the reliability of a software application considering the correlation between components. They used fewer parameters in their proposed method, while they eliminate many computational other methods by selectively characterizing the probability of components that contribute to system reliability. But the method proposed by Fiondella et al., Despite reducing software reliability calculations using the multivariate Bernoulli (MVB) distribution, could not use software architecture styles to reduce computing space.

Previous studies have not used the reliability evaluation of architectural styles based on the failure of correlated components or they have not used software architecture styles in the process of evaluating the reliability of software architecture based on correlated components. In their research, Franco [15] and Wang et al. [16] used the Markov model to evaluate the reliability of software architecture styles. They did not consider component correlation in the reliability evaluation process, so they require a lot of computation to evaluate the reliability of the software architecture. Franco et al. [15] exploited the formalism of architectural descriptive languages (ADLs) to automatically describe mathematical models to express the reliability behavior of a system. Then, by extending the concept of “automated evaluation”, he conducted a detailed analysis to identify the architectural weaknesses that are affecting the system. This analysis aims to provide information to architects about improving reliability and suggesting other options. Wang et al. [16] calculated the reliability of the software system by transforming software architecture styles to Markov models.

Also, Fiondella et al. [10] considered the correlation between components in evaluating software architecture and showed that their proposed method required less computational use of the multivariate Bernoulli (MVB) distribution and was therefore scalable, but did not use software architecture styles.

In this research, an efficient method is presented for evaluating the reliability of different software architectural styles
by considering the correlated component failures to achieve a more accurate result, so that the architect can be helped in selecting the appropriate style.

Since there are various types of architectural styles and no general and public classification in a way that arranges styles into a single framework, this study focuses on the general homogeneous styles including Batch-Sequential Style, Parallel style, Fault Tolerance Style, and Call and Return Style. The reason for choosing these styles is their adaptation properties with other styles and numerous additional architectural styles can be considered as the development of these four types. For example, the Hierarchically Heterogeneous Style and the Layered Style are similar to the Serial and Parallel styles, and the Client-Server Style is similar to the Call and Return Style [1].

In order to evaluate the software reliability in this study, the discrete-time Markov chain (DTMC) model [14] is used; in other words, according to the characteristics of software architecture and control flow between architectural components, the Markov model [17] is derived from the architecture styles and finally, the overall reliability concerning the correlated component failures will be calculated.

Because the proposed method uses architectural styles, the multivariate Bernoulli (MVB) distribution, correlated components, and fewer parameters to evaluate the reliability of the software architecture, the scalability of the system is improved because only the main components of the system are evaluated and the low-level components are not considered in the calculations.

The contributions of this paper are as follows:

- Using software architecture styles by considering correlations between components to reliability evaluation.
- Using Markov chains in the process of evaluating the reliability of software architectural styles based on correlated components
- Reducing calculations and improving the scalability of software architecture by considering the main components in architectural styles and explicit characterization of the joint distribution of system components.

The rest of the paper is organized as follows. Section 2 reviews the previous studies related to the topic. The third section presents a prediction method for reliability evaluation of homogeneous and heterogeneous software architecture styles with regards to correlated component failures. Section 4, contains the case study, and finally, section 5 includes the conclusions and future studies.

2. Literature Review

Reliability models are used for predicting, monitoring, and evaluating software reliability. Working on software reliability models began in the 1970s and the first model was introduced in 1972 [8]. Some studies are focused on reliability modeling during the testing phase [18]. One of these models is called the Black Box model [19], which considers the software as a black box and it models only interactions with the outside world, with no regard to its internal structure. Recent research efforts are focused on developing methods to predict the reliability of a software application that can be applied in the software architecture design phase.

Cheung et al. [20] used the discrete-time Markov chain model for the prediction of software reliability without using software architecture styles. This model calculates software reliability concerning components reliability. Cheung's model assumes that the failures of the components are independent and the reliability of each component is determined by the probability of proper functionality of the component.

Wang et al. [16] presented an analytical model for estimating the reliability of software architectural styles based on the discrete-time Markov chain model and without regard to the failure of the correlated components. In this model, according to the reliability of each component, operational features, and software architecture, the system architecture view turns into a state view. In fact, they are used the Cheung model for the four basic homogeneous and heterogeneous styles.

Kristiansen et al. [21] considered the effect of components dependencies in assessing the reliability of the system and used the discrete-time Markov chain's model for software modeling. They used direct calculation, Birnbaum measurement, and Principal Component Analysis (PCA) techniques for the selection of the most important components and the dependencies among them. This method is defined only for sequential or parallel software. They show the proposed method has more accurate results than an approach that assumes the components independently.
Fiondella et al. [22] presented an efficient algorithm to analyze the reliability of a software application according to the correlated component failures without using software architecture styles. The input of the algorithm is the component reliability and dependency matrix and the output is the probability of joint distribution of software components by using Poisson variables. They performed this technique on four different system architectures, and the results were compared with traditional methods that neglected correlation. They have concluded that system reliability with correlated component failures is higher than the software with independent components. The flaw of these methods is high and inefficient calculations, as such, they are not scalable and are not suitable for large software applications. Fiondella et al. [10] reformed their previous methods to provide a new approach that considers only the joint distribution of components that affect software reliability. Consequently, it is scalable and can be used in evaluating large systems.

Brosch et al. [23] expressed the drawbacks of existing methods that constraint the applicability and accuracy of reliability evaluation as follows:

- Many methods do not consider the effects of software operational profile (sequence of component calls and values of the input parameters) on the control diagram and data flow of software architecture.
- Many methods do not consider the effects of software’s execution environment on reliability. Because even if the software is error-free, again failure occurs for some reasons such as unavailability of hardware sources.
- Many methods use the Markov model in the modeling software system, which does not include many software engineering concepts (input parameters, interface descriptions, connectors, etc.).

To overcome these limitations, they are provided a new technique to evaluate the reliability of software architecture by using a Palladian Component Model (PCM) that is similar to the UML diagram. It involves all effective factors in the architecture such as the integration of all architectural aspects, software operational profile, and executing environment (taking into consideration the different situations of hardware sources availability) to increase the accuracy of reliability assessment.

Li et al. [24] provided an efficient approach for evaluating software reliability in regards to the failure of the correlated components that estimates the exact dependencies between components via the multivariate Bernoulli distribution (MVB) because most approaches used hypothetical data instead of measuring detailed data for coupling parameters of components. In this paper, a unified framework for measuring components correlation in the software will be built based on a comprehensive review of Object-oriented and procedure-oriented frameworks. Then various types of dependencies are determined based on these two frameworks. Finally, the software is modeled by DTMC.

Delac et al. [25] presented a method for improving the reliability of SOA-based applications. Service-Oriented Architecture (SOA) is an architectural style that provides strategies for the development of loosely coupled distributed systems. They are provided a method to improve the reliability of services in designing the most appropriate service composition with a focus on the reliability of critical components. This method includes Reliability Estimation, Weak Point Recommendation, and Weak Point Strengthening based on the Bayesian belief network.

Daraskhan Anjum et al. [26] analyzed the failure probability of software correlated components before implementation and detected the factors that cause software correlated failures in the Pakistan industry.

Aleti et al. [27] used Polynomial Chaos Expansion (PCE) as an exact method for uncertainty propagation and to estimate the performance of a software system. Experimental results on different case studies from different phases of software development show that their method can estimate the robustness of various performance indices accurately and rapidly.

Zhu et al. [28] proposed a non-homogeneous Poisson process (NHPP) software reliability model based on software fault dependency and imperfect fault removal. They defined two types of software faults and a two-phase debugging process for this model assuming that only Type 2 (dependent) software faults exist in Phase 2.

Bixin et al. [29] proposed an approach to evaluate software architecture evolution. In this process, they used sequential diagrams to model components interaction, a SPIN-based model checking to model and verify software architecture evolution. They first described the system as an automata model to represent the desired properties for verification. The model was then verified to see if it satisfied the properties in the state space. Cortellessa et al. [30] proposed
a model-driven approach to improving the availability of their software systems through refactoring actions. To evaluate the availability of software systems, they mapped UML models onto Generalized Stochastic Petri Nets (GSPN) analysis models and vice versa. They showed that their proposed method generates an analyzable availability model from the software architecture descriptions and a valid software architecture back model from the availability model. Sedaghatbaf et al. [31] introduce a SANAM model as a formal method for modeling software architectures and evaluating their quality attributes based on stochastic activity networks (SANs). They used hierarchical colored stochastic activity networks (HCSANs) for architecture modeling and used activity-marking oriented reward structures for evaluation of quality attributes such as security, dependability, and performance. Sedaghatbaf et al. [32] in another research introduce SQME as a framework for automatic evaluation of software architecture models. The framework uses evolutionary algorithms for architecture improvement, evidence theory for uncertainty handling, and EV/TOPSIS for making trade-off decisions. In their method, they considered attribute inter-dependency and presented an algorithm to transform the software architecture into a formal model, and demonstrated how their model could be used to evaluate reliability and security. Their proposed framework used standards such as UML to model software architecture and MARTE to add performance and time information to UML diagrams. Also, using the multi-criteria evolutionary algorithms available in SQME, the initial architectural model is iteratively modified and evaluated until Pareto-optimal models are found and finally, an EV/TOPSIS-based method was proposed to select the best model from the Pareto set. Also, in SQME, uncertainty in model parameters is expressed through intervals of possible values. In this study and their previous research, they considered the uncertainties in the model parameters. In [33], They used DAM profiles to determine the reliability parameters in the UML diagram. The resulting model was then transformed into a fault tree to evaluate reliability. Similar to [32] evidence theory is used to model the uncertainties in input parameters, propagate them through the fault tree and determine their impacts on the output measure. They use UML diagram for modeling software architectures and the DAM profile for specifying reliability parameters in the UML model. The constructed model is transformed into a fault tree for reliability evaluation. The experimental results show the estimated reliability bounds provide useful information for objective decision-making.

Ouhbi [34] performed a systematic mapping study to organize the existing software architecture evaluation approaches according to six classification criteria: research types, empirical types, contribution types, software quality models, quality attributes, and software architecture models. She has used 8 mapping questions for this classification.

Babar et al. [35] focused on achieving an evidence-based understanding of various aspects of software architecture evaluation activities, in particular improving the development of scenario profiles to describe desirable quality attributes. They presented a report on the findings of an empirical study of software architecture evaluation to evaluate the effectiveness of scenario development meetings in terms of the quantity of scenarios gained and lost during the evaluation process.

Milhem [36] introduced the architectural evaluation approach using implemented patterns and tactics to consider values of satisfaction of quality attributes (non-functional requirements). He extracted the implemented architectural patterns and tactics from a software architecture’s source code by Archie tools. Then he used the Goal-oriented Requirements Language (GRL) to documenting and modeling the patterns/tactics implemented by software architecture and their impact on quality attributes. Finally, he applied GRL/jUCMNav evaluation strategies to get the satisfaction values of the quality attributes.

Zhang et al. [37] mapped the software architecture model to the time-extended Petri net, which uses a state diagram to solve the problem and then used the state transition probability matrix to calculate the reliability of the entire system. They divided transitions into time transitions and instantaneous transitions in the time-extended Petri net model and introduced time-delay reliability and temporal reliability to time transition.

These researches have the following drawbacks:

1- Some of them have only paid attention to the evaluation of quality attributes at the architectural level. While to increase scalability, it is more appropriate to consider architectural design based on software architectural styles or patterns.
2- In most researches, the correlation between components is not used during the software architecture evaluation
process. While in the process of evaluating the reliability or security of software architecture, the failure of one component affects the other components.

3. Few studies have used the Markov model with the multivariate Bernoulli (MVB) distribution for evaluation. While the Markov model, like the Petri Net, shows well the different states between the components and the dependencies between them.

3. Method of Analysis

In this section, a scalable method for assessing the reliability of homogeneous and heterogeneous software architectural styles is proposed concerning the correlated component failures. Figure 1 shows the processes of the reliability evaluation method in homogeneous styles.

The abbreviations used are described at the end of this research.

**Step 1: Defining the architecture with a state diagram.** In the first step, the components of the software and the transmission of flow control between them are described by the state diagram which defines the dynamic behavior of the components.

**Step 2: Mapping the state diagram to the Markov model.** The state diagram is mapped to Markov’s model. This mapping can be a one-to-one or many-to-one. In other words, the states of several components may be interdependent while being executed, so they are mapped to one state in the Markov model (many to one mapping) and there is a possibility that the components states of the software are independent of each other, in that case, they are mapped in separate states (one to one mapping).

**Step 3: Creating the transition probability matrix of P and the transition matrix of M.** In the Markov model [17], the system will be placed in a specific state in each time period, and the transition from one state to another state changes randomly. To analyze this model, the probability of transition between different states must be calculated. If the Markov model has m state, a \( p_{ij} \) matrix is considered which each \( p_{ij} \) shows the probability of transfer from state \( i \) to state \( j \). It is worth mentioning that the probability of transfer between two states follows a Markov feature; that is, a transfer from \( S_i \) to \( S_j \) is merely dependent on the current state. The probability of transition between different states can be obtained in two ways [16]: (1) the operational profile may be used if available. (2) If it is assumed that the Markov model has \( m \) states and if in one state of this model there are more than one transfer to other states, thus, to find the probability of transition from state \( S_i \) to the state \( S_j \), two states will be considered according to Equations 1 as follows:

\[
  p_{ij} = \begin{cases} 
  \frac{t(i,j)}{\sum_{i=1}^{m} t(i,j)} & \text{if } i \geq 1, j \leq m \text{ and } s_i \text{ reach to } s_j \\
  0 & \text{otherwise}
  \end{cases}
\]

i. If a transition between the two states exists, the number of transfers from state \( i \) to state \( j \) to the total number of transfers that may occur from state \( i \) to other states should be calculated.

ii. If there is no transfer from state \( S_i \) to the state \( S_j \), then \( p_{ij} \) will be measured as zero in the matrix.

Transfer matrix of M in homogeneous styles based on the Wang model [16] is described as following. Where \( M(i,j) \) is the probability of successful transition of reaching state \( S_j \) from \( S_i \).

**Batch Sequential Style.** Since in this style at any moment, only one component is running so the control flow will be transferred to only one of its successors upon the completion of a component [1]. Assuming that the architecture is composed of \( K \) components and \( K \) states are in a Markov chain. Then, the transfer matrix of \( M \) can be obtained as follows:
\[ M(i, j) = \begin{cases} R_i P_j, & \text{s, can reach } s, \text{ for } i \geq 1, j \leq k \\ 0, & \text{s, can reach } s, \text{ for } i \geq 1, j \leq k \end{cases} \] (2)

where \( R \) represents the reliability of component \( C_i \).

**Parallel Style.** This style includes multiple components which are run concurrently. This will, therefore, reduce the service time required [1]. For \( k \) components, the transition matrix can be obtained as:

\[ M(i, j) = \begin{cases} R_i P_j, & s_j \not= s_i \\ \prod_{q=2}^{k} (1-R_q) R_i, & s_j \in s_i \\ 0, & s_j \text{ can not reach } s_i \end{cases} \] (3)

**Fault Tolerance Style.** This style is usually used to improve the availability of software systems. In addition, this style includes fault components which are a set of backup components and primary components. The primary and backup components are placed in parallel. Therefore, if one of these components fails, the backup components will still be able to provide the services and thus compensate the primary component failure. Moreover, if the new primary component fails, the replacement of the new primary component will be done by another backup component. This will be executed as long as a backup component is available to take over its responsibilities for the failure of the primary component [1]. Assuming there are \( k \) components, the transition matrix can be constructed as follows:

\[ M(i, j) = \begin{cases} R_i P_j, & s_j \not= s_i \\ \sum_{q=2}^{k} \prod_{n=1}^{q-1} (1-R_n) R_i, & s_j \in s_i \text{ and } s_j \text{ include } C_i \text{ to } C_w \\ 0, & \text{otherwise} \end{cases} \] (4)

**Call-and-return Style.** This style has been extensively used in the design of distributed systems. In the call and return style, the execution of one component usually requires the services provided by other components. Accordingly, a called component can be invoked by a calling component several times. Therefore, the execution of the called component can occur many times. Assuming there are \( k \) components, the transition matrix \( M \) can be constructed as follows:

\[ M(i, j) = \begin{cases} R_i P_j, & s_i \text{ can reach } s_j \\ P_j, & s_i \text{ can reach } s_j \text{, and } s_i \text{ is callee for } i \geq 1, j \leq k \\ 0, & \text{otherwise} \end{cases} \] (5)

**Heterogeneous styles.** For an architecture that is composed of heterogeneous styles, the transition matrix \( M \) can be computed by equation 6.

\[ M(i, j) = \begin{cases} 0, & s_i \text{ can not reach } s_j \\ P_j, & s_i \text{ is caller and } s_j \text{ is callee} \\ \mu P_j, & \text{otherwise} \end{cases} \] (6)

The calculation of \( \mu \) is done according to the type of style and number of running components.

**Step 4: Determining MVB parameters and MVB encoding.** In this step MVB [10] is performed on software components parameters (such as the reliability of components and dependency between them). Assume that the s-expected reliability of component \( i \) \( \{ E(R_i) \} \) is determined by a random variable of \( \mu_i \).
Correlations between the two components can be demonstrated by the correlation matrix of $\Sigma$ that each element $p_{ij}$ shows the correlation between the components of $i$ and $j$. Finally, MVB distribution is presented by MVB ($\mu, \Sigma$). Component’s reliability in MVB distribution is in the range $[0, 1]$ and correlations between two components are in the range $[-1, 1]$ [38]. These values can be derived from operational profiles [10]. Then, MVB components encode parameters as a function of independent Poisson variables $Y$. Encoding algorithm is as follows:

i. In the first iteration ($K = 1$), $\lambda_k$ is calculated according to Equation 8 and is shown as a matrix $\Lambda^i$ [10]. $\lambda_k$ is the parameter of the correlation encryption rate between $i$ and $j$. In fact, elements of the matrix which are the Poisson rates of $\lambda_k$, are calculated as a function of the MVB parameters ($\mu, \mu, \rho_j$) that $\rho_j$ is the correlation between components $i$ and $j$ [10].

\[
\lambda_k = \ln \left( 1 + \rho_j \sqrt{\frac{(1 - \mu_i)(1 - \mu_j)}{\mu_i \mu_j}} \right)
\]  

Since $\lambda_k = \lambda_k$, matrix $\Lambda$ is a symmetric matrix.

\[
\Lambda_{\text{sym}} = \begin{bmatrix}
\lambda_{11} & \lambda_{12} & \cdots & \lambda_{1s} \\
0 & \lambda_{22} & \cdots & \lambda_{2s} \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & \lambda_{ss}
\end{bmatrix}
\]  

In this step, the minimum value of the element $\lambda^k_{ij}$ is selected from the matrix $\Lambda^i$ and is considered the rate of the Poisson variable $Y_k$. Poisson variable $Y_k$ determines the correlation between the components of $i$ and $j$. If the value is equal to 1, components of $i$ and $j$ will fail. Then components of $i$ and $j$ will be added to set $S^i$, which consists of correlation components set.

ii. If the minimum amount of $\lambda^k_{ij}$ of the matrix $\Lambda^i$ could be subtracted from the other $\lambda_k$ of the matrix, where $i, j \in S^i$ and $i \leq j$, those components are added to the $S^i$ set.

iii. Matrix $\Lambda_{\text{sym}}^{i+}$ is obtained by subtracting the minimum element of $\lambda^k_{ij}$ of matrix elements of $\Lambda^i$ [22]. where $i, j \in S^i$ and $i \leq j$, according to Equation 9:

\[
\lambda^k_{ij} = \lambda^k_{ij} - \lambda^k_{\text{min}}
\]  

iv. If all matrix elements are equal to zero, go to Step 6. Otherwise, $K = k + 1$, and the encryption algorithm is repeated from Step 2. Finally, this process is continued until all the elements of the matrix $\Lambda$ are reduced to zero.

v. The result of the encryption algorithm is $K$ Poisson variables (in which zero represents the non-occurrence of the event (zero events) and lack of component failure) and $S^i$ set which will be displayed in a table format.
Then, based on the Poisson variable rate \( Y_k \) obtained from the encryption algorithm, zero event probability of Poisson variables is calculated according to Equation 10:

\[
x_i = P[Y_i = 0] = e^{-x_i}
\]

(10)

**Step 5: Estimating architecture reliability.** In this step, an efficient algorithm is used for software reliability evaluation which also considers the component’s correlation. This algorithm calculates only the probabilities of those combinations of components that are effective in the reliability according to the structure and architecture of the system as they assess the reliability of the system. The algorithm starts while assuming that all Poisson variables are zero events. The algorithm input is minimal cutset which is determined based on system architecture and the output is its tree structure. The purpose of this step is to find those combinations of Poisson variables that are effective in the reliability of the software. They should be considered in estimating the reliability of the system. The algorithm is as follows:

i. The algorithm starts with a state in which all Poisson variables \( Y \) are zero. This state forms the root node at the zero level of the tree.

ii. In the first iteration, the root node is produced \( m_i = n(n+1)/2 \) (n is the number of components) childs. In each of these \( m_i \) child nodes, a single distinct Poisson variable \( Y_i \) is set to 1 \( (1 \leq i \leq m) \). Then, for each node, the algorithm specifies a set of components that have failed due to \( Y_i = 1 \). If this set contains one or more components of the cutset, the node will be considered as a leaf of the tree.

iii. In the second iteration, after evaluating \( m_i \) nodes in a tree, subsets of nodes that do not lead to system failure are determined. For each of these nodes, child nodes are created. A child node is a combination of zero and one along with \( Y_i \) and \( Y_j \), where \( i < j \) and \( Y_i = 1 \) and \( Y_j = 1 \) specify the failed components set. Each of the nodes (failed components per node) is compared with the cutset. If it leads to system failure, the node will be removed from further consideration. Condition \( i < j \) ensures that each leaf is a distinct combination of the Poisson variables.

iv. The algorithm terminates if it fails to produce further nodes. The above steps will ensure that the algorithm produces and checks all effective combinations that contribute to system reliability.

v. At the end of the algorithm, nodes or combinations of Poisson variables that affect system reliability will be detected and will be mapped to the joint distribution of components.

Finally, the possibility of the joint distribution of components \( pr[R] \) will be calculated. Total calculated probabilities of one or more Poisson variables combinations which are mapped to a component joint distribution calculate the possibility of the component joint distribution. The Probability of Poisson variables \( Y \), with zero events is shown as \( x_i \) and the non-zero probability is shown by \( \overline{x_i} = 1 - x_i \). Since the Poisson variables are independent, the Poisson probability of each combination of variables can be calculated by \( x_i \) and \( \overline{x_i} \). Finally, Equation 11 is used to estimate the reliability of the system.

\[
E[A_{corr}] = \sum(E[A \mid R] \times pr[R])
\]

\[
pr[R] = x_1 \times x_2 \times \ldots \times x_{n(z+1)/2}
\]

(11)

where \( pr[R] \) is the possibility of component joint distribution and \( E[A \mid R] \) is the conditional software reliability.
estimate for each possible combination of operational and failed components. Figure 2 shows the state diagrams of the batch sequential, the parallel, the Fault-tolerance, and the Call and Return styles and their Markov models. Also, equation 13 is used to assess the reliability of each style, and table 1 shows the value of their parameters. Figure 3 shows the model steps for reliability evaluation of heterogeneous styles [16] in summary. As the figure shows, in the first step, the heterogeneous architecture of the system needs to be defined by a state diagram. In the second step, available styles in the software architecture are detected. Moreover, in heterogeneous architecture, architectural styles may have shared commonalities; i.e. a component(s) may belong to several different styles. At this stage, based on the number of architectural styles, separate sets will be considered which include components related to that style. If a software \((G)\) has \(x\) components and it shows each architectural component with \(C\alpha\), the following sets to separate the components of each style are defined as [16]:

- Set \(B\) is created for the Batch-Sequential style.
- Set \(P\) is created for the parallel style.
- Set \(F\) is created for the Fault tolerance style.
- Set \(C\) consists of the caller components that may call one or more components during their implementation.
- The set \(S\) is considered for callee components in the heterogeneous architecture.

It should be noted that if components belong to more than one particular style, it should be added to both sets that this creates a commonality between sets. In the next steps, state diagrams are mapped to the Markov model and they are combined with the method developed by Wang [16] to become a global Markov model. In the next step, the probability matrix \(P\) and the transfer matrix \(M\) based on the correlated component failures are created. In order to build transition matrix \(M\) in heterogeneous software architecture, Equation 6 is used. The \(\mu_i\) is calculated according to the styles and number of executing components in the state \((S_i)\) based on Equation 12.

\[
\mu_i = \begin{cases} 
\mu_c & \text{only } c \text{ is in } s, \\
1 - \prod (1 - \mu_s) & \forall c \text{ in } s \text{ and multiple fault tolerance component are executed} \\
\prod \mu_c & \forall c \text{ in } s \text{ and multiple parallel component are executed} 
\end{cases} 
\] (12)

Two final steps are similar to homogeneous styles.

Also, Table 2 shows the reliability value of the proposed method for different architecture styles in comparison to the wang [16] method.

4. Case study

In this section, a case study using the proposed method is evaluated. Assume that software architecture consists of eight components of \(C_1, C_2, \ldots, C_8\) [39]. The software comprised 3 styles of batch sequential architectural, call and return, and fault-tolerance. All required information about the architecture of the system is summarized in Table 3 [39].

The state diagram is depicted in Figure 4a. The components that are inside the dotted oval \((C_1\) and \(C_2\)) are fault tolerance style components. Components of \(C_1, C_2, \ldots, C_8\), and \(C_9\) have call and return relationships. Component \(C_1\) is the caller component, which may call \(C_2\) and \(C_3\) during their execution. Finally, \(C_{call}, C_1, \text{ and } C_2\) are batch sequential style components.

According to the above description, some components belong to more than one particular style. These components should be entered into set styles. For example, \(C_1\), as a caller component calls other components and it is executed sequentially. Therefore, it should be considered in both sets \(B\) and \(C\). Thus, set \(G\) is created for architecture and the components of each style are as follows:
Then the state diagram is mapped to the Markov model. To create Markov models in heterogeneous architecture, Markov models derived from the styles are combined. The integration between modes is done in the many-to-one mapping of the shared Markov models. The Markov system models are depicted in Figure 4b.

In the following, the transition probability of matrix $P$ for the Markov model is created. Also, to calculate the transition matrix $M$, the Markov model and Equation 6 are used. Since in state $S_2$, some Fault tolerance components are running, the calculation of $\mu_i$ is performed as follows by using Equation 12.

$$\begin{align*}
\mu_i &= 1 - \left[ (1 - \mu_i) \times (1 - \mu_i) \right] = 1 - \left[ (1 - 0.827) \times (1 - 0.781) \right] = 0.962 \\
p_{12} &= p_{\text{out1}} = p_{\text{out2}} = 1.00 \\
p_{23} &= p_{\text{out3}} = 1.00
\end{align*}$$

$$P = \begin{bmatrix}
0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0.22 & 0 & 0.11 & 0.22 & 0.45 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

$$M = \begin{bmatrix}
0 & 0.955 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0.962 & 0 & 0 & 0 & 0 \\
0 & 0.195 & 0 & 0.11 & 0.22 & 0.399 & 0 \\
0 & 0 & 0.918 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0.787 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0.944 \\
0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

The MVB parameters, including the reliability of the components and the correlation between components, are as follows:

$$R = \left\{ \mu_1 = 0.955, \mu_i = 0.962, \mu_i = 0.886, \mu_i = 0.918, \mu_i = 0.787, \mu_i = 0.944, \mu_i = 1.00 \right\}$$

$$\Sigma = \begin{bmatrix}
1.0 & 0.289 & 0.191 & 0.123 & 0.042 & 0.055 \\
1.0 & 0.331 & 0.067 & 0.101 & 0.102 \\
1.0 & 0.100 & 0.179 & 0.025 \\
1.0 & 0.078 & 0.149 \\
1.0 & 0.070 \\
1.0
\end{bmatrix}$$

In the following, the first repetition of the encryption algorithm is done and the matrix $\Lambda'$ is created.
The number of non-zero elements of the matrix $\Lambda'$ is 21, thus, 21 matrices of $\Lambda$ must be produced by the encryption algorithm until all elements become zero. The results of the encryption process are shown in Table 4, briefly. Each row represents an iteration, the minimum amount at each iteration matrix is considered as a Poisson variable rate of $Y_i$ and set $S^i$. Table 5 indicates the probability of zero events for Poisson variables $Y_i$, which is calculated according to $\lambda^i$ rate in Tables 4 and Equation 9.

In the proposed reliability assessment method based on the correlated component failure, minimal cutset in Figure 4b is $C = \{\{s_1\}, \{s_2\}, \{s_3\}, \{s_4\}\}$. To create the system tree, the root node is $Y = 000000000000000000000$ and then 21 child nodes for each Poisson variable of $Y_i$ are created. Table 6 shows the calculations of Poisson variables in the first level of the tree concerning cutset and Table 4.

According to Equations 11, the calculations of the reliability of the system regarding the failure of the correlated components are displayed in Table 7. With the continuation of calculations $E[A_{mm}] = 0.7422$ is obtained.

Table 8 shows the reliability values of the proposed method in comparison to some existing methods. According to the results in the table, the proposed method is more accurate than other methods. In Cheung's method [20] and Wang's method [16], the components are assumed to be independent. On the other hand, in the Wang method [16], software architecture styles are used. In the Fiondella method [10], the evaluation takes place according to the correlation between the components, regardless of the styles used in the architecture. The proposed method is a combination of these three methods that evaluates the reliability of software architecture by taking into account the correlation between the components on the architectural styles.

5. Conclusion

Since software architecture styles play an important role in satisfying quality attributes, in this research, software architecture styles transformed to a mathematical model based on the Markov model to evaluate reliability. Also, to increase the scalability and accuracy of the evaluation results, the correlation of the components was considered using the multivariate Bernoulli (MVB) distribution. Since in the multivariate Bernoulli (MVB) distribution only the components correlation matrix is created, therefore, only the components that affect the reliability of the software in the calculations are considered and therefore the proposed method can be used for large architectures as well. The evaluation results show the proposed method is more accurate than other methods. The evaluation of quality attributes such as performance and security in architectural styles can be raised as future works. Also, the effect of quality attributes on each other should be applied to software architecture styles to achieve more accurate results. In the continuation of this research, the correlation of components in other styles and patterns of software architecture such as service-oriented architecture and layered architecture should be used to evaluate quality attributes.

In this study, like many methods, basic information taking into consideration the reliability of components, the likelihood of transition between the components, and the correlation between components are available. But one of the available things to evaluate at the architectural level is to obtain the necessary information. In this study, a positive correlation between components and dependencies between components are assumed to be identical. Therefore, the proposed method can be generalized by considering negative correlations between the components and the exact relationship between the components.
The Abbreviation

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_{ij}$</td>
<td>Probability of transfer from state $i$ to state $j$</td>
</tr>
<tr>
<td>$M$</td>
<td>Transfer matrix</td>
</tr>
<tr>
<td>$m$</td>
<td>Number of states in Markov model</td>
</tr>
<tr>
<td>$S_i$</td>
<td>State $i$</td>
</tr>
<tr>
<td>$M(i,j)$</td>
<td>Probability of successful transition of reaching state $S_j$ from $S_i$</td>
</tr>
<tr>
<td>$t(i,j)$</td>
<td>Total number of invocations or control transfers from component $c_i$ to $c_j$</td>
</tr>
<tr>
<td>$R_i$</td>
<td>Reliability of component $C_i$</td>
</tr>
<tr>
<td>$E(R_i)$</td>
<td>s-expected reliability of component $i$</td>
</tr>
<tr>
<td>$\mu_i$</td>
<td>s-expected reliability of a component</td>
</tr>
<tr>
<td>$\lambda_{i,j}$</td>
<td>Rate parameter encoding correlation between Components $i$ and $j$</td>
</tr>
<tr>
<td>$\rho_{i,j}$</td>
<td>Correlation between components $i$ and $j$</td>
</tr>
<tr>
<td>$Y_k$</td>
<td>$K^{th}$ Poisson variable</td>
</tr>
<tr>
<td>$\Lambda^k$</td>
<td>$\Lambda$ matrix in iteration $k$</td>
</tr>
<tr>
<td>$S^i$</td>
<td>Set of components to which $Y^i \lambda^i_{\max}$ is added</td>
</tr>
<tr>
<td>$\lambda^i_{ij}$</td>
<td>$(i,j)$th entry of $\Lambda^i$</td>
</tr>
</tbody>
</table>

References


36- Bani Milhem, H. A. I. “Evaluating Software Architecture Based on Their Implemented Patterns and Tactics”, Doctoral dissertation, the University of Ottawa (2020).


*Sima Emadi* is an Assistant Professor and Director of Computer postgraduate at Computer Engineering Department, Islamic Azad University, Yazd Branch. He received the B.Eng. degree from Islamic Azad University, Iran, in 1995 and the M.S. degree from Islamic Azad University, Iran, in 1997, both in Computer Software engineering. In 2008 she completed the Ph.D. program at Islamic Azad University, Science and Research Branch, Iran. Her
current research interests include services computing Software, Web service Composition, Service Driven Architecture, Software Testing and Design Pattern.

Fig. 1. Processes of reliability evaluation method in homogeneous styles

Fig. 2. Architectural styles and their Markov models

Fig. 3. The model steps to evaluate the reliability in heterogeneous styles

Fig. 4. Description of system architecture

Table 1. Reliability calculation of each style
Table 2. The reliability value of the proposed method and the wang method
Table 3. Operating behavior of the system [39]
Table 4. Results of the encryption process of the MVB system
Table 5. The probability of zero events for Poisson variables
Table 6. The first tree level study system
Table 7. Calculation of software reliability
Table 8. Reliability values of the software in a variety of methods
Fig. 5. Processes of reliability evaluation method in homogeneous styles

- Defining the architecture with state diagram
- Mapping the state diagram to Markov model
- Creating the transition probability matrix of $P$ and the transition matrix of $M$
- Determining MVB parameters and MVB encoding
- Estimating architecture reliability
<table>
<thead>
<tr>
<th>Styles name</th>
<th>State model</th>
<th>Markov model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batch sequential</td>
<td><img src="https://via.placeholder.com/150" alt="Diagram" /></td>
<td><img src="https://via.placeholder.com/150" alt="Diagram" /></td>
</tr>
<tr>
<td>parallel</td>
<td><img src="https://via.placeholder.com/150" alt="Diagram" /></td>
<td><img src="https://via.placeholder.com/150" alt="Diagram" /></td>
</tr>
<tr>
<td>fault-tolerant</td>
<td><img src="https://via.placeholder.com/150" alt="Diagram" /></td>
<td><img src="https://via.placeholder.com/150" alt="Diagram" /></td>
</tr>
<tr>
<td>call and return</td>
<td><img src="https://via.placeholder.com/150" alt="Diagram" /></td>
<td><img src="https://via.placeholder.com/150" alt="Diagram" /></td>
</tr>
</tbody>
</table>

![Fig. 6. Architectural styles and their Markov models](https://via.placeholder.com/150)
Fig. 7. The model steps to evaluate the reliability in heterogeneous styles

1. Defining the heterogeneous behavior architecture with state diagram
2. Identifying basic styles heterogeneous architecture
3. Mapping the state diagram to Markov model
4. Integrating Markov models
5. Creating the transition probability matrix of $P$ and the transition matrix of $M$
6. Determining MVB parameters and MVB encoding
7. Estimating architecture reliability
(a) State diagram  

(b) Markov model

Fig. 8. Description of system architecture
Table 9. Reliability calculation of each style

| Style’s name          | Output | $\{E[A] | R \}$ | $pr\{r\}$ | $(E[A] | R ) \times pr\{R\}$ |
|-----------------------|--------|-----------------|----------|-----------------------------|
| Sequential style      | 1111   | 1.0000          | 0.7969   | 0.7969                      |
| Parallel              | 111    | 1.0000          | 0.7926   | 0.7926                      |
| fault tolerant        | 111    | 1.0000          | 0.8732   | 0.8732                      |
| call and return       | 1001   | 0.4000          | 0.0010   | 0.00040                     |
|                       | 1011   | 0.4000          | 0.0456   | 0.01824                     |
|                       | 1101   | 0.5263          | 0.0184   | 0.00968                     |
|                       | 1111   | 1.0000          | 0.7969   | 0.79690                     |

Table 10. The reliability value of the proposed method and the Wang method

<table>
<thead>
<tr>
<th>Name of styles</th>
<th>Proposed method</th>
<th>Wang method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batch sequential</td>
<td>0.7969</td>
<td>0.7669</td>
</tr>
<tr>
<td>parallel</td>
<td>0.7926</td>
<td>0.7668</td>
</tr>
<tr>
<td>Fault tolerance</td>
<td>0.8732</td>
<td>0.8536</td>
</tr>
<tr>
<td>Call and return</td>
<td>0.8252</td>
<td>0.7261</td>
</tr>
</tbody>
</table>

Table 11. Operating behavior of the system [39]

<table>
<thead>
<tr>
<th>Component</th>
<th>Reliability</th>
<th>Transition probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_{start}$</td>
<td>0.955</td>
<td>$p_{start} = 0.5, p_{start2} = 0.5$</td>
</tr>
<tr>
<td>$c_1$</td>
<td>0.827</td>
<td>$p_{s1} = 1.00$</td>
</tr>
<tr>
<td>$c_2$</td>
<td>0.781</td>
<td>$p_{s2} = 1.00$</td>
</tr>
<tr>
<td>$c_3$</td>
<td>0.886</td>
<td>$p_{s3} = 0.11, p_{s4} = 0.11, p_{s5} = 0.22, p_{s6} = 0.45$</td>
</tr>
<tr>
<td>$c_4$</td>
<td>0.918</td>
<td>$p_{s4} = 1.00$</td>
</tr>
<tr>
<td>$c_5$</td>
<td>0.787</td>
<td>$p_{s5} = 1.00$</td>
</tr>
<tr>
<td>$c_6$</td>
<td>0.944</td>
<td>$p_{read} = 1.00$</td>
</tr>
<tr>
<td>$c_{end}$</td>
<td>1.0</td>
<td></td>
</tr>
</tbody>
</table>
Table 12. Results of the encryption process of the MVB system

<table>
<thead>
<tr>
<th>repetition (k)</th>
<th>λ_{min}</th>
<th>Set s^i</th>
<th>Repetition (k)</th>
<th>λ_{min}</th>
<th>Set s^i</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0022</td>
<td>{1,2,3,4,5,6}</td>
<td>12</td>
<td>0.0009</td>
<td>{1,2}</td>
</tr>
<tr>
<td>2</td>
<td>0.0007</td>
<td>{1,2,4,5,6}</td>
<td>13</td>
<td>0.0079</td>
<td>{4,5}</td>
</tr>
<tr>
<td>3</td>
<td>0.0011</td>
<td>{1,2,3,4,5}</td>
<td>14</td>
<td>0.0088</td>
<td>{2,3}</td>
</tr>
<tr>
<td>4</td>
<td>0.0007</td>
<td>{1,2,3,5}</td>
<td>15</td>
<td>0.0118</td>
<td>{2}</td>
</tr>
<tr>
<td>5</td>
<td>0.0010</td>
<td>{3,4,5}</td>
<td>16</td>
<td>0.0242</td>
<td>{3,5}</td>
</tr>
<tr>
<td>6</td>
<td>0.0020</td>
<td>{2,5,6}</td>
<td>17</td>
<td>0.0296</td>
<td>{1}</td>
</tr>
<tr>
<td>7</td>
<td>0.0037</td>
<td>{2,3,5}</td>
<td>18</td>
<td>0.0488</td>
<td>{6}</td>
</tr>
<tr>
<td>8</td>
<td>0.0039</td>
<td>{5,6}</td>
<td>19</td>
<td>0.0661</td>
<td>{3}</td>
</tr>
<tr>
<td>9</td>
<td>0.0040</td>
<td>{1,3,4}</td>
<td>20</td>
<td>0.0663</td>
<td>{4}</td>
</tr>
<tr>
<td>10</td>
<td>0.0024</td>
<td>{3,4}</td>
<td>21</td>
<td>0.1921</td>
<td>{5}</td>
</tr>
<tr>
<td>11</td>
<td>0.0068</td>
<td>{1,2,3}</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 13. The probability of zero events for Poisson variables

<table>
<thead>
<tr>
<th>Y_1</th>
<th>x_1</th>
<th>Y_2</th>
<th>x_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y_3</td>
<td>0.9978</td>
<td>Y_{12}</td>
<td>0.9991</td>
</tr>
<tr>
<td>Y_4</td>
<td>0.9993</td>
<td>Y_{13}</td>
<td>0.9921</td>
</tr>
<tr>
<td>Y_5</td>
<td>0.9989</td>
<td>Y_{14}</td>
<td>0.9912</td>
</tr>
<tr>
<td>Y_6</td>
<td>0.9993</td>
<td>Y_{15}</td>
<td>0.9883</td>
</tr>
<tr>
<td>Y_7</td>
<td>0.9990</td>
<td>Y_{16}</td>
<td>0.9761</td>
</tr>
<tr>
<td>Y_8</td>
<td>0.9980</td>
<td>Y_{17}</td>
<td>0.9708</td>
</tr>
<tr>
<td>Y_9</td>
<td>0.9963</td>
<td>Y_{18}</td>
<td>0.9524</td>
</tr>
<tr>
<td>Y_{10}</td>
<td>0.9961</td>
<td>Y_{19}</td>
<td>0.9360</td>
</tr>
<tr>
<td>Y_{11}</td>
<td>0.9960</td>
<td>Y_{20}</td>
<td>0.9359</td>
</tr>
<tr>
<td>Y_{12}</td>
<td>0.9976</td>
<td>Y_{21}</td>
<td>0.8252</td>
</tr>
<tr>
<td>Y_{13}</td>
<td>0.9932</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 14. The first tree level study system

<table>
<thead>
<tr>
<th>Poisson variables combinations</th>
<th>Joint distribution of component</th>
<th>Reliability</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000000000000000000000</td>
<td>000000</td>
<td>No</td>
</tr>
<tr>
<td>010000000000000000000000</td>
<td>010000</td>
<td>No</td>
</tr>
<tr>
<td>001000000000000000000000</td>
<td>000001</td>
<td>No</td>
</tr>
<tr>
<td>000100000000000000000000</td>
<td>001001</td>
<td>No</td>
</tr>
<tr>
<td>000001000000000000000000</td>
<td>110001</td>
<td>No</td>
</tr>
<tr>
<td>000000100000000000000000</td>
<td>101100</td>
<td>No</td>
</tr>
<tr>
<td>000000010000000000000000</td>
<td>100101</td>
<td>No</td>
</tr>
<tr>
<td>000000001000000000000000</td>
<td>111100</td>
<td>No</td>
</tr>
<tr>
<td>000000000100000000000000</td>
<td>010011</td>
<td>No</td>
</tr>
<tr>
<td>000000000010000000000000</td>
<td>110011</td>
<td>No</td>
</tr>
<tr>
<td>000000000001000000000000</td>
<td>001111</td>
<td>No</td>
</tr>
<tr>
<td>000000000000100000000000</td>
<td>111001</td>
<td>Yes</td>
</tr>
<tr>
<td>000000000000010000000000</td>
<td>100111</td>
<td>No</td>
</tr>
<tr>
<td>000000000000001000000000</td>
<td>101111</td>
<td>No</td>
</tr>
<tr>
<td>000000000000000100000000</td>
<td>110101</td>
<td>No</td>
</tr>
<tr>
<td>000000000000000010000000</td>
<td>011111</td>
<td>No</td>
</tr>
<tr>
<td>000000000000000001000000</td>
<td>111110</td>
<td>No</td>
</tr>
<tr>
<td>000000000000000000100000</td>
<td>111011</td>
<td>No</td>
</tr>
<tr>
<td>000000000000000000010000</td>
<td>111101</td>
<td>Yes</td>
</tr>
<tr>
<td>000000000000000000001000</td>
<td>111101</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 15. Calculation of software reliability

| Output  | \( (E[A]|R) \) | \( pr\{r\} \) | \( (E[A]|R) \times pr\{R\} \) |
|---------|----------------|----------------|----------------------------------|
| 111001  | 0.5769         | 0.00632        | 0.00365                          |
| 111011  | 0.8036         | 0.04220        | 0.03391                          |
| 111101  | 0.6716         | 0.13040        | 0.08758                          |
| 111111  | 1.0000         | 0.61711        | 0.61711                          |

Table 16. Reliability values of the software in a variety of methods

<table>
<thead>
<tr>
<th>Method</th>
<th>System Reliability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cheung [20]</td>
<td>0.4787</td>
</tr>
<tr>
<td>Fiondella [10]</td>
<td>0.6191</td>
</tr>
<tr>
<td>Wang [16]</td>
<td>0.6428</td>
</tr>
<tr>
<td>Proposed method</td>
<td>0.7422</td>
</tr>
</tbody>
</table>