Scientia Iranica D (2022) 29(1), 135–149

SCIENTIA IRANICA

Sharif University of Technology
Scientia Iranica
Transactions D: Computer Science & Engineering and Electrical Engineering
http://scientiairanica.sharif.edu

Reliability evaluation of software architectural styles based on correlated component failure

S. Emadi*

Department of Computer Engineering, Yazd Branch, Islamic Azad University, Yazd, Iran.

Received 18 July 2020; received in revised form 3 July 2021; accepted 25 October 2021

KEYWORDS
Software architecture styles;
Reliability evaluation;
Correlated component failures;
Discrete-time Markov chain;
Quality attributes;
Software architecture design.

Abstract. This study aims to provide an efficient and scalable approach to evaluate the reliability of different software architectural styles concerning correlated components failures. To do so, a method based on the Discrete-Time Markov Chain (DTMC) model is proposed. In the proposed method, software architectural styles are used for reliability evaluation. The four main styles are transformed into Markov chain models and the transfer matrix is established for them; then, by using the Bernoulli distribution, the correlation between components is shown in the matrix and used in the evaluation process. The proposed method is scalable so that it can be used for large software architectures with heterogeneous and homogeneous styles. The evaluation results for the case study demonstrate that this method enjoys greater accuracy than other methods for the prediction of reliability of the software architectures. It is concluded that the proposed method is viable enough for a preliminary estimation of the software architecture reliability and can make a better comparison among various architectural styles to choose the most suitable one from the available options.

1. Introduction

Analysis, design, and implementation of software systems are intended for solving 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]. After determining the priority requirements, software architecture in the software development life cycle encapsulates 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 a significant role in satisfying the quality attributes of a software system, i.e. reliability, security, performance, availability, usability, and maintainability, its analysis and evaluation is essential to determining the level of satisfaction of these quality attributes based on the structure of and correlation between its components, because changing a design flaw, especially in large-scale software systems, at the implementation and testing phase is more expensive than that at the architectural design stage [3].

However, designing a software architecture is a

*. *Tel.: 035-31872707*
E-mail address: emadi@iauyazd.ac.ir

doi: 10.24200/sci.2021.56444.4726

complex activity throughout the software development. In the process of determining the components and the relationships between them, the non-functional requirements (quality attributes) of the software must be considered and software architectures be established based on quality attributes.

Architectural styles and patterns provide a way to organize software components so that the designer can easily build the software and satisfy customer functional and non-functional requirements. Software architectural 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.

Bachmann et al. [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 an architecture, because architectural styles define the corresponding elements and the rules of how they interact [5].

Therefore, choosing an appropriate architectural style is one of the important decisions in software architectural design [6]. In addition to the description of the software and its decomposition into components, software architectural styles have a major impact on qualitative attributes of designed software (e.g., reliability, performance, and 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 of styles enables software designers to make design decisions more accurate and select an 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 in 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] claimed that the most contemporary analysis approaches were not scalable because they required a large number of parameters and were computationally inefficient and these approaches required 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 was calculated based on the reliability knowledge of individual components and the communication between them. In [13], the probability of joint distribution was 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 fewer parameters and performs efficient computation is essential to analyzing the reliability of software based on a large scale that also considers the correlated component failures. Therefore, they presented 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 eliminated many other computational methods by selectively characterizing the probability of components that contribute to system reliability. However, the method proposed by Fiondella et al., despite reducing software reliability calculations using MVB distribution, could not use software architecture styles to reduce the computing space.

Previous studies have not conducted the reliability evaluation of architectural styles based on the failure of correlated components nor have they not employed software architectural 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] employed the Markov model to evaluate the reliability of software architectural styles. They did not consider component correlation in the reliability evaluation process; therefore, they require much computation to evaluate the reliability of the software architecture. Franco [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", they conducted a detailed analysis to identify the architectural weaknesses affecting the system. This analysis aims to provide architects with information about improving reliability and suggesting other options. Wang et al. [16] calculated the reliability of the software system upon transforming software architectural styles to Markov models.

Moreover, Fiondella et al. [10] considered the correlation between components in evaluating software architecture and demonstrated that their proposed method required less computational use of the MVB distribution and was therefore scalable, but did not use software architectural 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 would feel aided in selecting the appropriate style.

Since there are different types of architectural styles with 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, MVB distribution, correlated components, and fewer parameters to evaluate the reliability of the software architecture, the scalability of the system is enhanced 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 architectural styles for reliability evaluation by considering correlations between components;
- Using Markov chains in the process of evaluating the reliability of software architectural styles based on correlated components;
- Reducing calculational burden and enhancing 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 this study is organized as follows. Section 2 reviews the previous studies related to the topic. Section 3 presents a predictive method for reliability evaluation of homogeneous and heterogeneous software architectural styles with regard 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 have 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 the interactions made only with the outside world, regardless of its internal structure. Recent research efforts have targeted developing methods for predicting the reliability of a software application that could be applied at the software architecture design phase.

Cheung [20] applied the discrete-time Markov chain model for the prediction of software reliability without using software architectural 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 regardless of 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 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 model for software modeling. They used direct calculation, Birnbaum measurement, and Principal Component Analysis (PCA) techniques to select the most important components and the dependencies among them. This method is defined only for sequential or parallel software. They demonstrated that the proposed method had more accurate results than the approach that assumes the components independently.

Fiondella [22] presented an efficient algorithm to analyze the reliability of a software application according to the correlated component failures without software architectural styles. The input of the algorithm includes the component reliability and dependency matrix; the output includes the probability of joint distribution of software components using Poisson's variables. The above researchers performed this technique on four different system architectures, and the results were compared with those of traditional methods that neglected correlation. They concluded that system reliability with correlated component failures was higher than the software with independent components. The flaw of these methods is high and inefficient calculations; 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 affecting 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 constrain the applicability and accuracy of reliability evaluation as follows:

i. 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;

ii. Many methods do not consider the effects of software's execution environment on reliability. Because even if the software is error-free, failure occurs again for some reasons such as unavailability of hardware sources;

iii. 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, a new technique is provided to evaluate the reliability of software architecture using a Palladian Component Model (PCM), which 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 execution of the environment (taking into consideration different situations of hardware sources availability) to increase the accuracy of reliability assessment.

Li et al. [24] provided an efficient approach to evaluating software reliability in regard to the failure of the correlated components that estimates the exact dependencies between components via the multivariate Bernoulli distribution because most approaches use 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 is formed based on a comprehensive review of object-oriented and procedure-oriented frameworks. Then, different 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 provided a method to enhance 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.

Anjum and Mustafa [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 estimated the performance of a software system. Experimental results of different case studies from different phases of software development illustrate that their method can estimate the robustness of various performance indices accurately and rapidly.

Zhu and Pham [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 at Phase 2.

Li et al. [29] proposed an approach to evaluate software architecture evolution. In this process, they used sequential diagrams to model components interaction, that is, an 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 demonstrated that their proposed method generated an analyzable availability model from the software architecture descriptions and a valid software architecture back model from the availability model.

Sedaghatbaf and Azgomi [31] introduced 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. In another research, Sedaghatbaf and Azgomi [32] introduced 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; in addition, they 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, upon 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 is 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 the related previous research, the authors considered the uncertainties in the model parameters. In [33], DAM profiles were used to determine the reliability parameters in the UML diagram. The resulting model was then transformed into a fault tree to evaluate reliability. Similar to the case of [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 used 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 that 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 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 in the evaluation process.

Bani 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) for documenting and modeling the patterns/tactics implemented by software architecture and their impact on quality attributes. Finally, the author employed 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 that uses a state diagram to solve the problem and then, applies 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. To increase scalability, it is more appropriate to consider an architectural design based on software architectural styles or patterns;
2. In most researches, the correlation between components is not used in the software architecture evaluation process. However, 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 MVB distribution for evaluation. However, 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

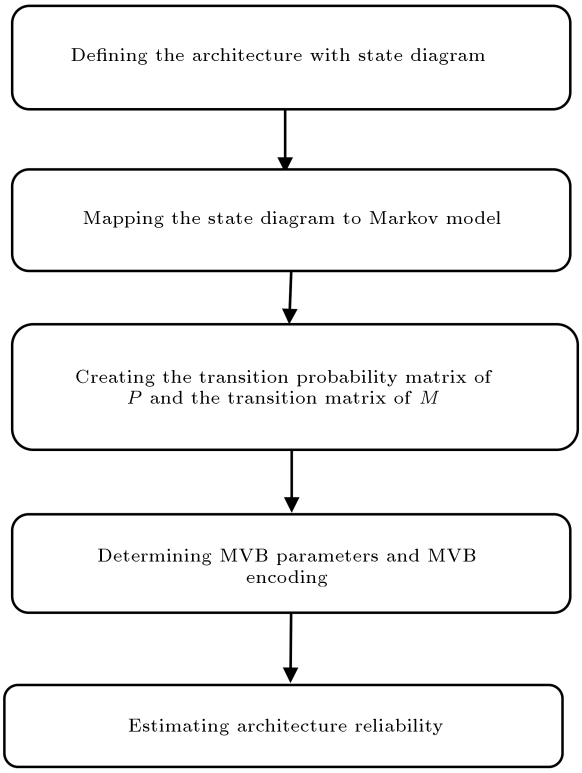


Figure 1. Processes of reliability evaluation method in homogeneous styles.

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 either one-to-one or many-to-one. In other words, the states of several components may be interdependent while being executed; thus, they are mapped to one state in the Markov model (many-to-one mapping) and there is a possibility that the component states of the software are independent of each other in which case they are mapped in separate states (one-to-one mapping);

Step 3: Creating a transition probability matrix of $\boldsymbol{P}$ and a transition matrix of $\boldsymbol{M}$. In the Markov model [17], the system is 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 $p_{m\times m}$ matrix is considered where each p_{ij} shows the probability of transfer from state i to state j. It is worth mentioning that the probability of transfer between the 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 there are more than one transfer to other states in one state of this model, then to find the probability of transition from the state S_i to the state S_j, two states will be considered according to Eq. (1) as follows:

$$p_{ij} = \begin{cases} t(i,j)/\sum\limits_{n=1}^{m} t(i,j) & S_i \text{ reach to } S_j \\ & \text{for } i \geq 1,\ j \leq m \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

i. If a transition between the two states exists, the ratio of 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 the state S_i to state S_j, then p_{ij} is measured as zero in the matrix.

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

Batch Sequential Style. Since only one component is running in this style at any moment, the control flow is transferred to only one of its successors upon the completion of a component [1]. Assuming that the architecture is composed of K components with K states in a Markov chain, the transfer matrix of M can be obtained as follows:

$$M(i,j) = \begin{cases} R_i P_{ij} & S_i \text{ can reach } S_j \text{ for } i \geq 1,\ j \leq k \\ 0 & S_i \text{ can reach } S_j \text{ for } i \geq 1,\ j \leq k \end{cases} \tag{2}$$

where R_i represents the reliability of component C_i.

Parallel style. This style includes multiple components that are run concurrently. This will, therefore, reduce the service time required [1]. For k components, the transition matrix can be obtained as follows:

$$M(i,j) = \begin{cases} R_i P_{ij} & S_i \notin S_p \\ \prod R_n P_{nj} & S_i \in S_p \\ 0 & S_i \text{ cannot reach } S_j \end{cases} \tag{3}$$

Fault tolerance style. This style is usually used to improve the availability of software systems. In addition, this style includes fault components that represent 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 are still able to provide the services and thus, overcome 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 that there are k components, the transition matrix can be constructed as follows:

$$M(i,j)=\begin{cases} R_iP_{ij} & S_i \notin S_b \\ R_{a1}+\sum\limits_{q=a2}^{ar}\left[\prod\limits_{m=a1}^{q-1}(1-R_m)R_n\right] & \\ \quad S_i \in S_b \text{ and } S_i \text{ include } C_{a1} \text{ to } C_{ar} & \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

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 may occur many times. Assuming that there are k components, the transition matrix M can be constructed as follows:

$$M(i,j)=\begin{cases} R_iP_{ij} & S_i \text{ can reach } S_j \\ p_{ij} & S_i \text{ can reach } S_j \text{ and } S_j \text{ is callee} \\ & \text{for } i \geq 1,\ j \leq k \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

Heterogeneous styles. For an architecture composed of heterogeneous styles, the transition matrix M can be computed through Eq. (6).

$$M(i,j)=\begin{cases} 0 & S_i \text{ cannot reach } S_j \\ p_{ij} & S_i \text{ is caller and } S_j \text{ is callee} \\ \mu_i p_{ij} & \text{otherwise} \end{cases} \tag{6}$$

The calculation of μ_i is done according to the type of style and the number of running components.

Step 4: Determining MVB parameters and MVB encoding. In this step, MVB [10] is carried out for software component 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 μ_i:

$$\mu_i = R_iP_{ij} \qquad \text{from matrix } M. \tag{7}$$

Correlations between the two components can be demonstrated by the correlation matrix of $\sum$ where each element p_{ij} shows the correlation between the components i and j. Finally, MVB distribution is presented by MVB $(\mu, \sum)$. Component's reliability in MVB distribution ranges between [0, 1] and the correlations between the two components are in the range of [-1, 1] [38]. These values can be derived from operational profiles [10]. Then, MVB components encode parameters as a function of independent Poisson variable Y. Encoding algorithm is as follows:

i. In the first iteration ($K = 1$), λ_{ij} is calculated according to Eq. (8) and is shown as a matrix Λ^1 [10]. λ_{ij} is the parameter of the correlation encryption rate between i and j. In fact, elements of the matrix, including Poisson rates of λ_{ij}, are calculated as a function of the MVB parameters $(\mu_i, \mu_j, \rho_{ij})$ in which ρ_{ij} is the correlation between the components i and j [10]:

$$\lambda_{ij} = \ln\left(1+\rho_{ij}\sqrt{\frac{(1-\mu_i)(1-\mu_j)}{\mu_i\mu_j}}\right). \tag{8}$$

Since $\lambda_{ji} = \lambda_{ij}$, matrix Λ is a symmetric matrix.

$$\Lambda_{n\times n} = \begin{bmatrix} \lambda_{11} & \lambda_{12} & \cdots & \lambda_{1n} \\ 0 & \lambda_{22} & \cdots & \lambda_{21} \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & \lambda_{nn} \end{bmatrix}.$$

In this step, the minimum value of the element λ_{ij}^k is selected from the matrix Λ^k and is considered as the rate of the Poisson variable Y_K. Poisson variable Y_K determines the correlation between the components i and j. If the value is equal to 1, the components i and j fail. Then, the components i and j are added to set S^k, which consists of a set of correlation components;

ii. If the minimum value of λ_{ij}^k in the matrix Λ^k is subtracted from the other λ_{ij} values in the matrix, where $i, j \in S^k$ and $i \leq j$, those components are added to the S^k set;

iii. Matrix Λ^{k+1} is obtained by subtracting the minimum element of $\lambda_{i,j}^k$ from the matrix elements of Λ^k [22] where $i, j \in S^k$ and $i \leq j$, according to Eq. (9):

$$\lambda_{ij}^{k+1} = \lambda_{ij}^k - \lambda_{\min}^k. \tag{9}$$

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 continues until all the elements of the matrix Λ are reduced to zero;

v. The result of the encryption algorithm includes K Poisson variables (in which zero represents the non-occurrence of the event (zero events) and lack of component failure) and S^k set to be displayed in a table format.

Then, based on the Poisson variable rate (Y^k) obtained from the encryption algorithm, zero event probability of Y_i Poisson variables [22] is calculated according to Eq. (10):

$$x_i = P_r\{Y_i = 0\} = e^{-y_i}. \tag{10}$$

Step 5: Estimating architecture reliability. In this step, an efficient algorithm is utilized for software reliability evaluation that considers the component's correlation. This algorithm calculates only the probabilities of those combinations of components that are effective in reliability according to the structure and architecture of the system as they assess the reliability of the system.

The algorithm begins while assuming that all Poisson variables are zero events. The algorithm input is minimal cut-set that 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 given 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 $m_1 = n(n+1)/2$ (n is the number of components) as child. In each of these m_1 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 is considered as a leaf of the tree;

iii. In the second iteration, after evaluating m_1 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 $i < j$, $Y_i = 1$, 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 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 mapped to the joint distribution of components.

Finally, the possibility of the joint distribution of components ($Pr\{R\}$) is calculated. Total calculated probabilities of one or more Poisson variable combinations, which are mapped to a component joint distribution, are employed to calculate the possibility of the component joint distribution. The probability of Poisson variables Y_i with zero events is denoted by x_i and the non-zero probability is denoted by $\overline{xi} = 1 - xi$. Since the Poisson variables are independent, the Poisson probability of each combination of variables can be calculated by xi and $\overline{xi}$. Finally, Eq. (11) is used to estimate the reliability of the system:

$$E\lfloor A_{\text{corr}} \rfloor = \sum (E\lfloor A \rfloor |R) \times pr\{R\},$$

$$Pr\{R\} = X_1 \times X_2 \times \cdots \times X_{n\times(n+1)/2}, \tag{11}$$

where $Pr\{R\}$ is the possibility of component joint distribution and $(E[A]|R)$ 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, Eq. (3) is used to assess the reliability of each style, and Table 1 shows the values of their parameters.

Table 1. 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

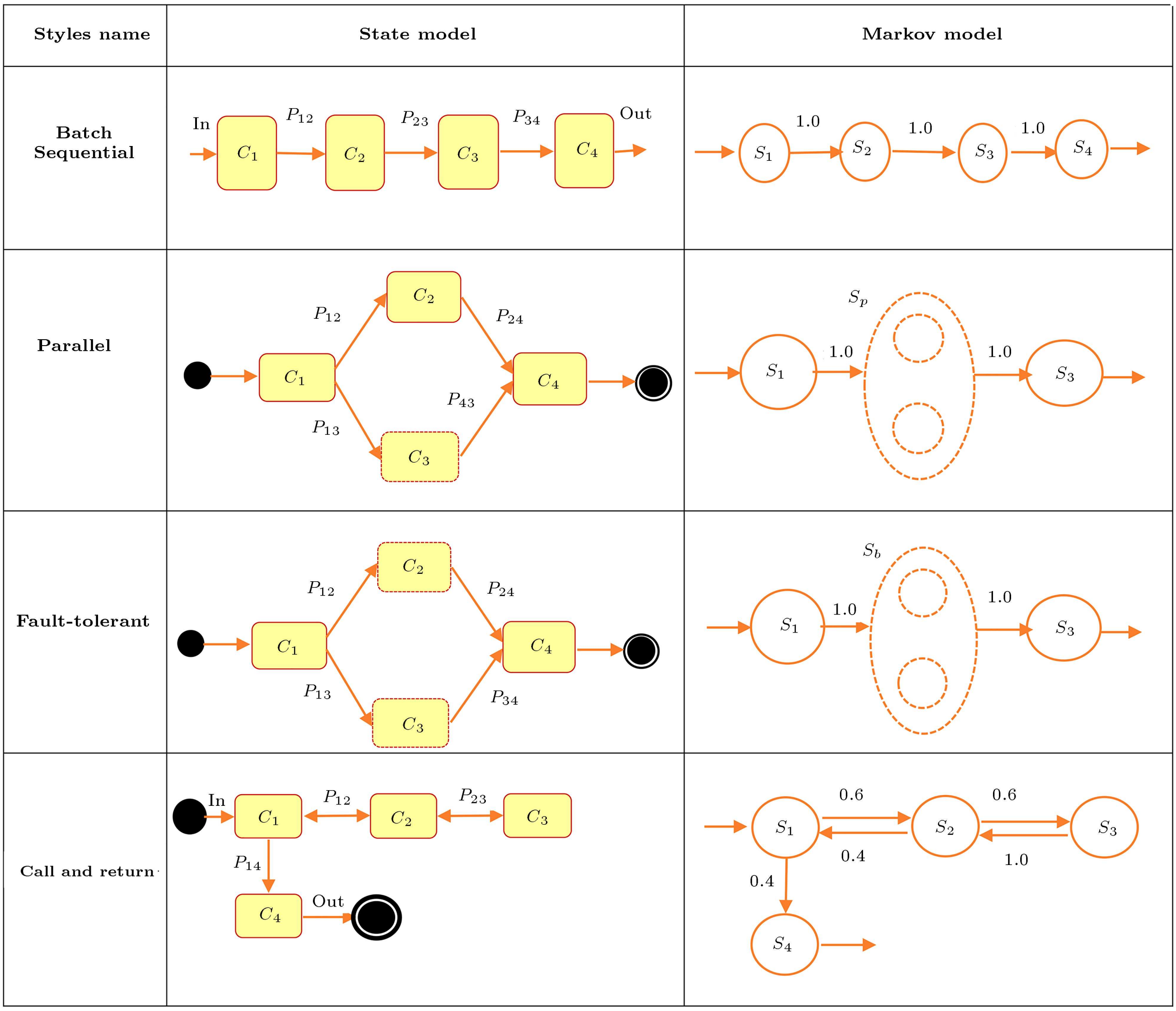


Figure 2. Architectural styles and their Markov models.

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 are considered that include components related to that style. If a software (G) has x components and indicate each architectural component with C_a, the following sets to separate the components of each style are defined as follows [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;
- Set S is considered for the 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 create 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 the heterogeneous software architecture, Eq. (6) is used. μ_i is measured according to the styles and number of components used in the state

$$\mu_i = \begin{cases} \mu_\alpha & \text{only } c_\alpha \text{ is in } s_i \\ 1-\prod\limits_{\alpha}(1-\mu_\alpha) & \forall\, c_{\alpha i} \text{ in } s_i \text{ and multiple fault tolerance component are executed} \\ \prod\limits_{\alpha} \mu_\alpha & \forall\, c_{\alpha i} \text{ in } s_i \text{ and multiple parallel component are executed} \end{cases} \quad (12)$$

Box I

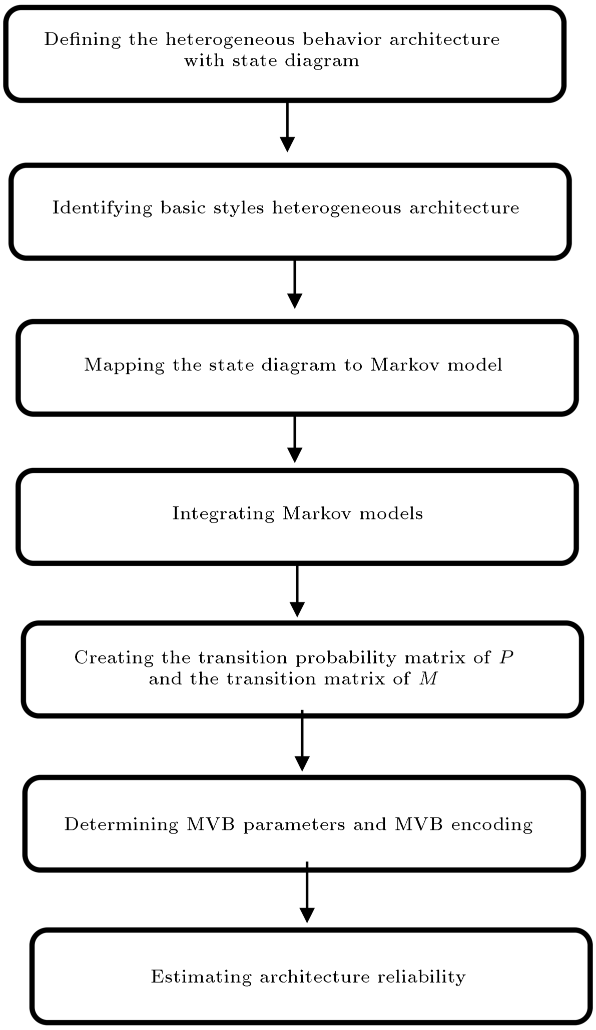


Figure 3. The model steps to evaluate the reliability in heterogeneous styles.

(S_i) based on Eq. (12) as shown in Box I. The 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's [16] method.

4. Case study

In this section, a case study using the proposed method is evaluated. Assume that the software architecture consists of eight components of $C_1, C_2, \cdots, C_8$ [39].

Table 2. The reliability value of the proposed method and Wang's method.

Name of styles	Proposed method	Wang method
Batch sequential	0.7969	0.7669
Parallel	0.7926	0.7668
Fault tolerance	0.8732	0.8536
Call and return	0.8252	0.7261

The software comprised 3 styles: batch sequential architectural, call-and-return, and fault tolerance. All the required information about the architecture of the system is summarized in Table 3 [39].

The state diagram is depicted in Figure 4(a). The components that are inside the dotted oval (C_1 and C_2) are fault tolerance style components. Components of C_3, C_4, and C_5 are characterized by call-and-return relationships. Component C_3 is the caller component, which may call C_4 and C_5 during their execution. Finally, C_{start}, C_3, and C_6 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_3 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 given below:

$$G = \{C_\alpha | component \;\; C_\alpha \in B \cup F \cup C \cup S\},$$

$$B = \{C_{start}\} \cup \{C_3\} \cup \{C_6\} = \{C_{start}, C_3, C_6\},$$

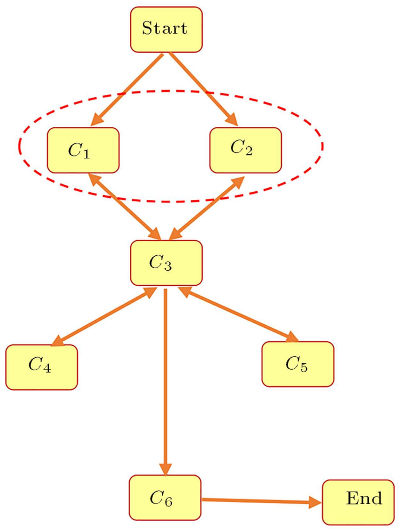
$$F = \{C_1, C_2\} \quad C = \{C_3\} \quad S = \{C_4, C_5\}.$$

Then, the state diagram is mapped to the Markov model. To create Markov models in the 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 4(b).

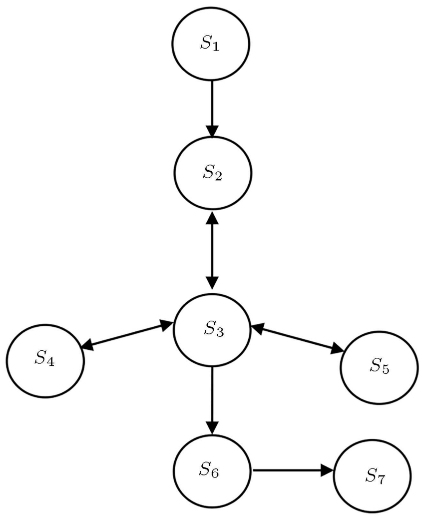
In the following, the transition probability of matrix P for the Markov model is created. Also,

Table 3. Operating behavior of the system [39].

Component	Reliability	Transition probability
c_{start}	0.955	$p_{\text{start1}} = 0.5$, $p_{\text{start2}} = 0.5$
c_1	0.827	$p_{13} = 1.00$
c_2	0.781	$p_{23} = 1.00$
c_3	0.886	$p_{31} = 0.11$, $p_{32} = 0.11$, $p_{34} = 0.11$, $p_{35} = 0.22$, $p_{36} = 0.45$
c_4	0.918	$p_{43} = 1.00$
c_5	0.787	$p_{53} = 1.00$
c_6	0.944	$p_{6\text{end}} = 1.00$
c_{end}	1.0	



(a) State diagram



(b) Markov model

Figure 4. Description of system architecture.

to calculate the transition matrix M, the Markov model and Eq. (6) are used. Since some fault-tolerant components are running in state S_2, the calculation of μ_i is performed using Eq. (12) as follows.

$$\mu_2 = 1 - [(1-\mu_1) \times (1-\mu_2)]$$

$$= 1 - [(1-0.827) \times (1-0.781)] = 0.962,$$

$$p_{12} = p_{start\ 1} = p_{start\ 2} = 1.00,$$

$$p_{23} = p_{13} = 1.00,$$

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

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

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

$$R = \langle \mu_1 = 0.955, \mu_2 = 0.962, \mu_3 = 0.886,$$

$$\mu_4 = 0.918, \mu_5 = 0.787, \mu_6 = 0.944, \mu_7 = 1.000 \rangle,$$

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

Table 4. Results of the encryption process of the MVB system.

Repetition (k)	$\lambda^k_{\min}$	Set s^k	Repetition (k)	$\lambda^k_{\min}$	Set s^k
1	0.0022	{1,2,3,4,5,6}	12	0.0009	{1,2}
2	0.0007	{1,2,4,5,6}	13	0.0079	{4,5}
3	0.0011	{1,2,3,4,5}	14	0.0088	{2,3}
4	0.0007	{1,2,3,5}	15	0.0118	{2}
5	0.0010	{3,4,5}	16	0.0242	{3,5}
6	0.0020	{2,5,6}	17	0.0296	{1}
7	0.0037	{2,3,5}	18	0.0488	{6}
8	0.0039	{5,6}	19	0.0661	{3}
9	0.0040	{1,3,4}	20	0.0663	{4}
10	0.0024	{3,4}	21	0.1921	{5}
11	0.0068	{1,2,3}			

Table 5. The probability of zero events for Poisson variables.

Y_k	x_k	Y_k	x_k
Y_1	0.9978	Y_{12}	0.9991
Y_2	0.9993	Y_{13}	0.9921
Y_3	0.9989	Y_{14}	0.9912
Y_4	0.9993	Y_{15}	0.9883
Y_5	0.9990	Y_{16}	0.9761
Y_6	0.9980	Y_{17}	0.9708
Y_7	0.9963	Y_{18}	0.9524
Y_8	0.9961	Y_{19}	0.9360
Y_9	0.9960	Y_{20}	0.9359
Y_{10}	0.9976	Y_{21}	0.8252
Y_{11}	0.9932		

In the following, the first repetition of the encryption algorithm is done and the matrix Λ^1 is created:

$$\Lambda^1 = \begin{bmatrix} 0.0460 & 0.0124 & 0.0148 & 0.0079 & 0.0047 & 0.0029 \\ & 0.0387 & 0.0233 & 0.0040 & 0.0104 & 0.0049 \\ & & 0.1210 & 0.0107 & 0.0329 & 0.0022 \\ & & & 0.0856 & 0.0120 & 0.0108 \\ & & & & 0.2395 & 0.0088 \\ & & & & & 0.0576 \end{bmatrix}.$$

The number of non-zero elements of the matrix Λ^1 is 21; thus, 21 matrices of Λ 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 value in each iteration matrix is considered as a Poisson variable rate of Y_k and set S^k. Table 5 indicates the probability of zero events for Poisson variables Y_i, which is calculated according to $\lambda^{k_{\min}}$ rate in Table 4 and Eq. (9).

Table 6. The first tree level study system.

Poisson variables combinations	Joint distribution of component	Reliability
100000000000000000000	000000	No
010000000000000000000	001000	No
001000000000000000000	000001	No
000100000000000000000	000101	No
000010000000000000000	110001	No
000001000000000000000	101100	No
000000100000000000000	100101	No
000000010000000000000	111100	No
000000001000000000000	010011	No
000000000100000000000	110011	No
000000000010000000000	000111	No
000000000001000000000	001111	No
000000000000100000000	111001	Yes
000000000000010000000	100111	No
000000000000001000000	101111	No
000000000000000100000	110101	No
000000000000000010000	011111	No
000000000000000001000	111110	No
000000000000000000100	110111	No
000000000000000000010	111011	Yes
000000000000000000001	111101	Yes

In the proposed reliability assessment method, based on the correlated component failure, minimal cutset in Figure 4(b) is $C = \{\{s_1\}, \{s_2\}, \{s_3\}, \{s_6\}\}$. 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 calculation of Poisson variables at the first level of the tree with respect to cutset and considering Table 4.

Table 7. 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 8. Reliability values of the software in a variety of methods.

Method	System reliability
Cheung [20]	0.4787
Fiondella [10]	0.6191
Wang [16]	0.6428
Proposed method	0.7422

According to Eqs. (11), the calculation of the reliability of the system regarding the failure of the correlated components is given in Table 7. As a result of the calculations, $E[A_{\text{corr}}] = 0.7422$ 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 [20] and Wang's methods [16], the components are assumed independent. On the other hand, in the case of Wang's method [16], software architectural styles are used. In the case of Fiondella's 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, they were transformed into 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 only the components correlation matrix was created in the MVB distribution, only the components that affect the reliability of the software in the calculations were considered and, therefore, the proposed method could be used for large architectures, as well. The evaluation results point to the greater accuracy of the proposed method than other methods.

The evaluation of quality attributes such as performance and security in architectural styles can be raised for the 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 between the 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 on the reliability of components, the likelihood of transition between the components, and the correlation between components is available. However, essential information is necessary to be obtained for the purpose of further evaluation. In this study, a positive correlation between components and dependencies between components were 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.

Nomenclature

p_{ij}	Probability of transfer from state i to state j
M	Transfer matrix
m	Number of states in Markov model
S_i	State i
$M(i,j)$	Probability of successful transition of reaching state S_j from S_i
$t(i,j)$	Total number of invocations or control transfers from component c_i to c_j^i
R_i	Reliability of component C_i
$E(R_i)$	s-expected reliability of component i
μ_i	s-expected reliability of a component
$\lambda_{i,j}$	Rate parameter encoding correlation between components i and j
$\rho_{i,j}$	Correlation between components i and j
Y_k	Kth Poisson variable
Λ^k	Λ matrix in iteration k
S^k	Set of components to which $Y^k \lambda^k_{\min}$ is added
λ^k_{ij}	(i,j)th entry of Λ^k

References

1. Aghaee Ghazvini, G. and Emadi, S. "An analytical algorithm of component-based heterogeneous software architectural styles performance prediction", *Journal of Advances in Computer Research*, **5**(3), pp. 85–100 (2014).

2. Ingeno, J., *Software Architect's Handbook: Become a Successful Software Architect by Implementing Ef-*

fective Architecture Concepts, Packt Publishing Ltd (2018).

3. Wang, W.L., Wu, Y., and Chen, M.H. "An architecture-based software reliability model", *International Symposium on Dependable Computing*, Hong Kong, pp. 143–150 (1999).

4. Bachmann, F., Bass, L., Clements, P., et al., *Documenting Software Architectures: Views and Beyond*, Second Edn., Addison-Wesley (2010).

5. Dobrica, L. and Niemela, E. "A survey on software architecture analysis methods", *IEEE Transactions on Software Engineering*, **28**(7), pp. 638–653 (2002).

6. Clements, P.C. "Coming attractions in software architecture", In *Proc. the Joint Workshop on Parallel and Distributed Real-Time Systems*, Geneva, Switzerland, pp. 2–9 (1997).

7. Zeng, B., Li, C., and Liu, S.F. "A novel grey target decision-making model based on cobweb area and its application for choosing the software development pattern", *Scientia Iranica. Transaction E, Industrial Engineering*, **23**(1), p. 361 (2016).

8. Caiuta, R., Pozo, A., and Vergilio, S.R. "Meta-learning based selection of software reliability models", *Automated Software Engineering*, **24**(3), pp. 575–602 (2017).

9. Ghafarian Salehi Nezhad, A., Eshraghniaye Jahromi, A., Salmani, M.H., et al. "A four-phase algorithm to improve reliability in series-parallel systems with redundancy allocation", *Scientia Iranica*, **21**(3), pp. 1072–1082 (2014).

10. Fiondella, L., Rajasekaran, S., and Gokhale, S.S. "Efficient software reliability analysis with correlated component failures", *IEEE Transactions on Reliability*, **62**(1), pp. 244–255 (2013).

11. Fiondella, L., Rajasekaran, S., and Gokhale, S.S. "Efficient system reliability with correlated component failures", *IEEE 13th International Symposium on High-Assurance Systems Engineering (HASE)*, Boca Raton, FL, USA, pp. 269–276 (2011).

12. Singpurwalla, N.D., *Reliability and Risk: A Bayesian Perspective*, John Wiley & Sons (2006).

13. Littlewood, B. "The impact of diversity upon common mode failures", *Reliability Engineering & System Safety*, **51**(1), pp. 101–113 (1996).

14. Bindal, D. "A review of Markov model for estimating software reliability", *International Journal of Advanced Research in Computer Science and Software Engineering*, **3**(6), pp. 426–433 (2013).

15. Franco, J.M.D.C.S. "Automated reliability prediction and analysis from software architectures", Doctoral Dissertation, University of Coimbra (2016).

16. Wang, W.L., Pan, D., and Chen, M.H. "Architecture-based software reliability modeling", *Journal of Systems and Software*, **79**(1), pp. 132–146 (2006).

17. Fazlollahtabar, H. and Jalali, S.G. "Adapted Markovian model to control reliability assessment in multiple AGV", *Scientia Iranica*, **20**(6), pp. 2224–2237 (2013).

18. Tyagi, K. and Sharma, A. "Reliability of component-based systems: A critical survey", *ACM SIGSOFT Software Engineering Notes*, **36**(6), pp. 1–6 (2011).

19. Gokhale, S.S. and Trivedi, K.S. "Analytical models for architecture-based software reliability prediction: A, unification framework", *IEEE Transactions on Reliability*, **55**(4), pp. 578–590 (2006).

20. Cheung, R.C. "A user-oriented software reliability model", *IEEE Transactions on Software Engineering*, **2**, pp. 118–125 (1980).

21. Kristiansen, M., Winther, R., and Natvig, B. "Component dependencies in compound software", *International Journal of Reliability, Quality and Safety Engineering*, **17**(05), pp. 465–493 (2010).

22. Fiondella, L. "Joint distribution decomposition for the reliability analysis of systems with correlated failures", In *Proc. of ISSAT International Conference on Reliability and Quality in Design*, San Francisco, CA, pp. 275–279 (2009).

23. Brosch, F., Koziolek, H., Buhnova, B., et al. "Architecture-based reliability prediction with the palladio component model", *IEEE Transactions on Software Engineering*, **38**(6), pp. 1319–1339 (2012).

24. Li, X., Yin, Y., Fiondella, L., et al. "Software reliability analysis considering correlated component failures with coupling measurement framework", *Journal of Systems Engineering and Electronics*, **26**(5), pp. 1114–1126 (2015).

25. Delac, G., Silic, M., and Srbljic, S. "A reliability improvement method for SOA-based applications", *IEEE Transactions on Dependable and Secure Computing*, **12**(2), pp. 136–149 (2015).

26. Anjum, D. and Mustafa, T. "Reliability of software correlated components failure in pakistan industry", *International Journal of Information Engineering and Electronic Business*, **9**(2), p. 27 (2017).

27. Aleti, A., Trubiani, C., van Hoorn, A., et al. "An efficient method for uncertainty propagation in robust software performance estimation", *Journal of Systems and Software*, **138**, pp. 222–235 (2018).

28. Zhu, M. and Pham, H. "A two-phase software reliability modeling involving with software fault dependency and imperfect fault removal", *Computer Languages, Systems & Structures*, **53**, pp. 27–42 (2018).

29. Li, B., Liao, L., and Yu, X. "A verification-based approach to evaluate software architecture evolution", *Chinese Journal of Electronics*, **26**(3), pp. 485–492 (2017).

30. Cortellessa, V., Eramo, R., and Tucci, M. "From software architecture to analysis models and back: Model-driven refactoring aimed at availability improvement", *Information and Software Technology*, **127**, 106362 (2020).

31. Sedaghatbaf, A. and Azgomi, M.A. "Software architecture modeling and evaluation based on stochastic activity networks", In *International Conference on Fundamentals of Software Engineering*, Tehran, Iran, pp. 46–53 (2015).

32. Sedaghatbaf, A. and Azgomi, M.A. "SQME: a framework for modeling and evaluation of software architecture quality attributes", *Software & Systems Modeling*, **18**(4), pp. 2609–2632 (2019)

33. Sedaghatbaf, A. and Azgomi, M.A. "Reliability evaluation of UML/DAM software architectures under parameter uncertainty", *IET Software*, **12**(3), pp. 236–244 (2018).

34. Ouhbi, S. "Software architecture evaluation: A systematic mapping study", *13th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE)*, Funchal, Madeira Portugal, pp. 447–454 (2018).

35. Babar, M.A., Shen, H., Biffl, S., and Winkler, D. "An empirical study of the effectiveness of software architecture evaluation meetings", *IEEE Access*, **7**, pp. 79069–79084 (2019).

36. Bani Milhem, H.A.I. "Evaluating software architecture based on their implemented patterns and tactics", Doctoral dissertation, the University of Ottawa (2020).

37. Zhang, C., Ma, Y., Wang, X., et al. "Software architecture modeling and reliability evaluation based on Petri net", In *2017 IEEE International Conference on Dependable Systems and Their Applications (DSA)*, Beijing, China, pp. 51–56 (2017).

38. Prentice, R.L. "Binary regression using an extended beta-binomial distribution, with discussion of correlation induced by covariate measurement errors", *Journal of the American Statistical Association*, **81**(394), pp. 321–327 (1986).

39. Kamavaram, S. and Goseva-Popstojanova, K. "Sensitivity of software usage to changes in the operational profile", In *28th Annual NASA Goddard Software Engineering Workshop*, Greenbelt, MD, USA, pp. 157–164 (2003).

Biography

Sima Emadi is an Assistant Professor and the Director of Computer postgraduate at the Computer Engineering Department, Islamic Azad University, Yazd Branch. She received the BEng degree from Islamic Azad University, Iran in 1995 and the MS degree from Islamic Azad University, Iran in 1997, both in Computer Software Engineering. In 2008, she completed the PhD 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.