The Journal of Systems & Software 202 (2023) 111726

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

An empirical study of software architecture resilience evaluation methods[☆]

Jiaxin Pan, Zixuan Liu, Donglin Li, Lulu Wang, Bixin Li [*]

School of Computer Science and Engineering, Southeast University, Nanjing, 211189, China
The College of Software Engineering, Southeast University, Nanjing, 211189, China

ARTICLE INFO

Article history:
Received 19 December 2022
Received in revised form 9 April 2023
Accepted 23 April 2023
Available online 29 April 2023

Keywords:
Software architecture resilience
Empirical study
"4+1" view model
Evaluation methods

ABSTRACT

Resilience is one of the most essential quality properties of software systems, the resilience of software architectures plays an important role in the security of a software system. However, even though there are some methods have been proposed for evaluating the resilience of software architecture in the past few years, most of them are validated only by case studies with some specific application scenarios. We do not find a work which has provided a wide empirical verification and comparison of these different methods. To fill this gap, we explore and compare five typical software architecture resilience evaluation methods by experiments in this paper, and try to find which methods are better in which aspects. We have obtained the following findings: first, the five methods studied in this paper are effective and consistent in the trend of resilience change; secondly, the change of architecture resilience is actually related to the specific attributes and component relationships in the architecture; finally, systems designed in an object-oriented style are generally more resilient than most other design styles studied.

1. Introduction

Resilience was originally used in the field of physics as the attribute that describes a material's ability to absorb external energy and recover from the effects of outside forces (Hutchison and Sterbenz, 2009). When used in the context of software systems, the term "resilience" usually refers to a system's ability to maintain an acceptable level of service in the face of various faults and challenges, as well as the ability to recover to a normal operational state with all available resources (Ahmed and Hussain, 2007). Software architecture resilience is one of the most important quality properties of software systems, as it measures how well a software system can provide its service in a stable and continuous manner, especially for those who require a high level of service stability and security and need to function effectively in the face of external interference or attacks, such as streaming media, e-commerce platforms, and cloud services companies. Software resilience evaluation at the architectural level is a crucial method for ensuring software quality from the earliest stages of software design as part of a comprehensive software resilience evaluation. Therefore, we will focus on empirical study of the methods for evaluating the software architecture resilience in this paper.

Since software architecture resilience evaluation is a vital part of software quality assurance from the outset of program design, it must be incorporated into the software development process. There are a number of researchers have already proposed some architecture-based resilience evaluation methods. For examples, the Markov Modeling method was once used in modeling self-adaptive systems and thus evaluating system resilience in 2013 (Cámara et al., 2013, 2014), the attack surface method was used to evaluate network systems' resilience (Zhang et al., 2021); additionally, there are also some architecture resilience evaluation methods based on Petri Network (Smith et al., 2011; Maza and Megouas, 2021) or Bayes Network (Liu et al., 2017; Hosseini et al., 2014; Cai et al., 2018; Sanjay et al., 2022). However, there still has some shortcomings with existing methods: ① *some of the methods have only been tested on specific cases and lack sufficient validity tests*; ② *there is no unified definition of resilience, both connotation and denotation of resilience are different from each other*; ③ *there is not enough comparative studies of different methods.*

Therefore, on the basis of existing researches on software architecture resilience evaluation, we will study and compare in depth several typical resilience evaluation methods in order to fill the aforementioned gap. We first summarize and examine the similarities and differences between these methods, and then the methods will be verified and compared through experiments. We have designed two sets of experiments of architecture evolution

☆ Editor: W. Eric Wong.
* Corresponding author at: School of Computer Science and Engineering, Southeast University, Nanjing, 211189, China.
E-mail address: bx.li@seu.edu.cn (B. Li).

https://doi.org/10.1016/j.jss.2023.111726
0164-1212/© 2023 Elsevier Inc. All rights reserved.

scenarios and a set of comparative experiments between systems with different architecture styles. Through these experiments, we will compare and verify the effectiveness of five resilience evaluation methods. In addition, we also identify some key factors that possibly affect the resilience of software architecture, and summarize some architectural styles that are generally considered more resilient.

The remainder of the paper is structured as follows: Section 2 provides some contexts and background knowledge about our empirical study; Section 3 summarizes current several representative resilience evaluation methods; Section 4 discusses the implementation of five typical methods; Section 5 mainly focuses on the research questions and experimental analysis; and Section 6 concludes our work and discusses some limitations and potential research directions.

2. Background knowledge

The definition of software architecture resilience used in our empirical study will be discussed first in this section, then the "4+1" view model will be introduced in detail, and finally why the"4+1" model is used as the input of our prototype implements and experiments will be discussed in this section too.

2.1. Software architecture resilience

There are some definitions of resilience in the current papers (Ford et al., 2013; Bishop et al., 2011; Holling, 1973; Ferris, 2019; Liu et al., 2010, 2018; Cimellaro et al., 2010), however they are always a little bit difference between them, some concentrate on *safety* and *reliability*, some concentrate on *availability* and *restoration*, and others may be concentrate on *robustness* and *rapidity* etc.

To realize the suitable comparison, we provide a unified definition: the software architecture resilience used in this paper is defined as follows: *The ability of SA to maintain and restore software system functions and missions after the software system is attacked.* And resilience includes six sub-attributes: *reliability, restoration, availability, safety, robustness and rapidity*.

- *Reliability* means the ability of the system to maintain system functions after being attacked (Hosseini et al., 2014).
- *Restoration* means the ability of the system to recover from a damaged state to a normal state (Hosseini et al., 2014; Cai et al., 2018).
- *Availability* is the proportion of time a system is in a workable state (Cai et al., 2018).
- *Safety* is the ability of the system to resist attacks (Zhang et al., 2021; Theisen et al., 2018; Manadhata and Wing, 2011).
- *Robustness* describes the ability of the system to withstand risk shocks and disturbances before it is damaged (Zhu et al., 2019).
- *Rapidity* is defined as the speed with which a system can recover from a damaged state to a normal state (Zhu et al., 2019).

In this paper, all six of these sub-attributes are used to evaluate software architecture resilience. But different sub-attributes have different emphasis: *safety* and *robustness* are used to evaluate the ability of software to resist attacks; both *reliability* and *availability* are used to evaluate the ability of software to maintain functions after an attack; both *restoration* and *rapidity* are used to evaluate the ability of software to recover to normal state after being attacked, *restoration* focuses on the level of software recovery, and *rapidity* focuses on the speed of software recovery to the normal state.

2.2. "4+1" View model

The "4+1" architectural model (Kruchten, 1995) is a view model that outlines the design of a software-intensive system based on concurrent multiple views. These views define the system from many stakeholder perspectives. Besides, different views only contain some features of the system, while ignoring irrelevant features. The "4+1" view model has five view models: *logical view model*, *development view model*, *process view model* and *physical view model*. Moreover, we may additionally define the architecture using use-cases or scenarios, this is the fifth view model, and we regard it as *scenario view model*.

- *Logical view model* focuses on functional requirements — what types of services should the system provide to users. Beginning with the issue domain, the system is dissected, and a set of important abstract representations of the system is obtained.
- *Development view model* emphasizes the real arrangement of program modules. Software is divided into several little components that may be built by a single programmer or a small team.
- *Process view model* focuses on non-functional requirements such as performance, availability and so on.
- *Physical view model*, also known as the deployment view, is used to map software to hardware and solve system topology, system installation, communication and other problems.
- *Scenario view model* describes the relationship between system participants and functional use-cases, and reflects the interaction design and final requirements of the system.

In this paper, the "4+1" model is used as a tool for describing software architecture models because it can adequately describe various aspects of software architecture models from multiple perspectives: the process view focuses on the dynamic processes and state changes in the actual use of the system, and the development view focuses on the logical relationships and component structures between the different components and modules of the software system. When evaluating the resilience of a system architecture, we need to consider not only properties related to the static component relationships of the system (e.g. reliability, robustness), but also properties related to dynamic state changes and recovery of the system in actual operation (e.g. restoration, rapidity), so we use the *process view model* (similar to activity diagram in UML Specification) and the *development view model* (similar to component diagram in UML Specification) as input to each method. Of course, we also use other view models when we compute the sub-properties of resilience, we mean that these other view models are *logical view model*, *physical view model*, and *scenario view model* etc. in our method.

3. An overview of architecture resilience evaluation methods

We summarize some existing resilience evaluation methods and list them in Table 1, where we summarize seven typical methods from following several aspects: the source, basic theory of resilience, connotation of resilience, application scope, advantages and disadvantages.

3.1. Attack surface

Attack surface in software architecture refers to the subset of software system which can be used by an attacker to attack the system, the entry and exit points of a system, the channels, and the untrusted data items together make up the attack surface of

Table 1
An overview of current resilience evaluation methods.

Method	Related theory	Resilience definition	Application	Advantages	Disadvantages
Network architecture resilience evaluation based on attack surface analysis (Zhang et al., 2021)	Attack surface analysis	Safety, Restoration	Network Systems	Focus on the possible attacks on the system, and give design and repair suggestions.	Focus on static analysis, lack of dynamic evaluation.
System resilience evaluation method based on static Bayesian network (Hosseini et al., 2014)	Bayesian network	Reliability, Restoration	Engineered systems	Be able to statistically represent resilience in compact causal graphs.	Do not provide feedback loops.
System resilience evaluation method based on dynamic Bayesian network (Cai et al., 2018)	Dynamic Bayesian network	Availability, Restoration	Availability-based engineering	Combine static analysis and dynamic simulation.	A lot of computing resources are required for calculation.
Modeling and evaluation of system resilience based on minimal path (Proag, 2014; Li et al., 2016)	Minimal path	Robustness, rapidity, availability	Process system	The modeling process is clear, the performance can be described using minimal path, and various metrics of resilience can be evaluated.	The modeling process can only be modeled for process-based diagrams
Architecture-based software reliability modeling (Cámara et al., 2013)	Markov model	Reliability, restoration	Software architecture model	Can describe the transitions of the system between different states through Markov chains efficiently.	It is difficult to model the situation after a system is attacked.
Multidimensional simulation for resilience evaluation of complex systems (Gama Dessavre et al., 2016)	System Simulation Modeling	Reliability, Vulnerability, Survivability, Recoverability	Complex Systems	Evaluate the resilience from multidimensional functions	It is difficult to build a performance model over time based on the "4+1" model.
Architecture reliability analysis based on Petri net (Liu et al., 2017)	Petri net	Reliability	Cloud Data Systems	The Petri modeling method is relatively general and has a strong ability to describe the model.	Petri modeling requires a lot of actual data support

a software system (Manadhata and Wing, 2011; Theisen et al., 2018).

Attack surface was once widely used in software security evaluation because it describes a potentially attacked subset of a system. But in subsequent research, attack surfaces have also been shown to be applicable to system resilience assessment: in 2021, attack surface analysis was used to assess the resilience of network systems (Zhang et al., 2021). Using the attack surface of a system to assess the resilience of software focuses on the system's ability to resist attacks and to maintain and recover its functionality after an attack, in other words, the security and reliability of the system.

3.2. Static Bayesian network

Yodo and Wang et al. proposed a method for evaluating the system resilience based on the static Bayesian network in 2016 (Hosseini et al., 2014). The method defines system resilience as a function of two essential properties: *reliability* and *restoration*. Static Bayesian network is used as the evaluation tool to model and measure resilience under the influence of internal and external disturbances. Bayesian network is built based on *system specific characteristics*, so it needs a lot of prior knowledge and manual operations. This method can be applied to the resilience evaluation of various *engineering systems*, such as Electric Motor Supply Chain and Production Process.

3.3. Dynamic Bayesian network

Cai and Xie et al. proposed a system resilience evaluation method based on dynamic Bayesian network in 2017 (Cai et al., 2018). Resilience is viewed as an inherent characteristic of a system. Resilience consists of two sub-attributes, *performance* and *time-related* attributes. External factors such as attack and interference are not used as criteria for resilience evaluation. This method divides the system structure into three categories: *serial*, *parallel* and *voting*, and constructs the Bayesian network according to these three structures. Then, according to the *recovery rate* and *failure rate* of each node, the node state transition probability between adjacent time slices is calculated, so that a dynamic Bayesian network is obtained. By calculating the availability of each time slice, we can get a set of availability that changes with time, and then calculate the resilience value according to the resilience calculation formula. This method can be applied to the resilience evaluation of *Availability-based engineering*, such as nine-bus power grid system.

3.4. Markov Modeling

Markov Modeling method means using a stochastic model to describe a system with Markov properties in probability theory (Gagniuc, 2017). When used in software architecture evaluation, Markov method usually refers to the method of modeling a system using a Markov model and then studying and evaluating the Markov model. To model a system by Markov methods, we first need to assume that the architecture system conforms to two important assumptions of Markov models: firstly, the operation of each state the system is in depends only on the previous state; and secondly, that the transition probability matrix between states of the system does not change over time. The Markov model was applied to the evaluation of the reliability of software architectures as early as 2005 (Wang et al., 2006). In 2013 the Markov method was applied to software architecture resilience evaluation (Cámara et al., 2013), and after that there was also an empirical study about how to use Markov method to evaluate self-adaptive system's resilience (Cámara et al., 2014).

3.5. Minimal path

Li et al. proposed a system resilience evaluation method based on the minimal path in 2016 (Li et al., 2016). They argue that resilience refers to the ability of the system to quickly recover from disturbances. They proposed the concept of system resilience limit and the definition of system resilience. The system resilience is then modeled and evaluated using the *minimal path*. Finally, the network system is verified by *Monte Carlo simulation* (Li et al., 2016). Among them, the minimum path refers to the path from the source point to the sink point in the diagram, and the same node in the path will not be traversed multiple times (Locks, 1978). This method can be applied to various *process systems*.

3.6. Multidimensional simulation

To compare the system resilience among systems (or different modifications to a system), by introducing a new dimension to system resilience models, called stress, to mimic the definition of resilience in material science (Gama Dessavre et al., 2016). Modeling and comparing different attack types and resilience evaluation criteria through multidimensional functions, but it is difficult to build a performance model over time based on the "4+1" model.

3.7. Petri network

A novel colored generalized stochastic petri net (CGSPN) model based on IT infrastructures is build up, which clearly reflects the dynamic behavior and service request processing procedure under the active–active mechanism (Liu et al., 2017), the service reliability based IT infrastructure can be evaluated, while reliability is a key part of system resilience.

4. Implementation of five methods

We can find from Table 1 that seven typical methods have been used to evaluate the resilience of software system and some of them are likely to be used to evaluate the resilience of software architecture. In fact, some of them have provided a software architecture based method for evaluating system or network and showed their effectiveness (Cámara et al., 2014).

In this paper, we will do a comprehensive empirical study for five representative methods, including *attack surface analysis*, *static Bayesian network*, *dynamic Bayesian network*, *minimal path*, and *Markov model*. All of them show that they have excellent ability in the assessment of system resilience. However, few researches focused on the evaluation of software architecture resilience using these methods. Therefore, we want to find which methods are better when we use then to evaluate software architecture resilience in this paper.

Although we mentioned seven methods in the previous section, there are two methods that we do not cover in the implementation and experimentation section, they are *multidimensional simulation method* and *petri network method*. For the first one, it is difficult to model software performance over time while we are modeling the system at the architecture level; and for the second method, because the petri network mainly focuses on the only one sub-attribute of resilience, and requires a lot of real data of the system we want to evaluate, which is very difficult when we only have a scratch or an architecture level model, and currently we do not have UML cases for petri network.

To do better comparison and analysis, we will implement these five methods for evaluating software architecture resilience, where we use the "4+1" view model as input, that means we use "4+1" view models to model software architecture, and we use the definition of resilience in Section 2.A as the basic definition of software architecture resilience.

4.1. Implementation of attack surface analysis method

Attack surface analysis can be implemented at different levels, from the software system code to the software architecture level, in 2013, a method to assess the attack surface with the help of the system's SysML activity diagram was presented (Ouchani and Lenzini, 2014). The attack surface-based architecture resilience evaluation in this paper also starts from the activity diagram of the system: firstly based on the system's activity diagram, we detect the attack surface and corresponding components inside the system architecture model. Then according to the CAPEC database, we generate possible attacks based on the types of nodes inside the attack surface (Ouchani and Lenzini, 2014). Finally combining the attack probability of possible attacks and the reliability of each node, we find the average possibility of each node to sustain or recover from certain attacks in the activity diagram, and that is the final resilience assessed by our method, more formally: Let N be the activity diagram of the given architecture model, A be the set of nodes in N which are inside the detected attack surface, the resilience of the system can be calculated by (1):

$$resilience = \frac{\sum_{n \in N} prob(n)}{|N|} \tag{1}$$

The function $prob()$ denotes the chance of a node sustain or recover from attacks that may happen on it (2).

$$prob(node) = \frac{\sum_{a \in A} p(a) * restoration(node)}{|A|} \tag{2}$$

If this node is not in attack surface, that means this node cannot be directly attacked by outside attackers, so the *prob* denotes its reliability (3).

$$prob(node) = reliability(node) \tag{3}$$

4.2. Implementation of static Bayesian network method

To apply the static Bayesian network method to the "4+1" architecture model and reduce manual operations, we analyze the *component diagram* to determine the dependency of each component, and then construct a Bayesian network based on these relationships.

The operating status of components affects reliability and restoration. We utilize the Bayesian network to transmit the probability of different nodes to the reliability and restoration nodes, and then calculate the resilience value.

To construct a Bayesian network, we must study the system component dependencies and classify them as follows:

- Components that are directly attacked;
- Components that are not affected by any other components or attacks;
- Components affected by one or more components;
- Components that affect reliability;
- Components that affect restoration.

According to the classification of components and the Bayesian network structure and calculation principle, we can obtain the calculation formula of resilience, which is Eq. (4). This formula represents the probability of the system maintaining resilience, and the value range is 0 to 1. The lower the value, the worse the resilience of the system.

$$\begin{aligned} P(resilience = True) = {} & \sum P(attacks) \\ & \times P(components\ not\ affected\ by\ any\ others) \\ & \times P(components\ directly\ attacked|\ attacks) \\ & \times P(components\ affected\ by\ one\ or\ more\ components \end{aligned}$$

$$| \; components \; affecting \; this \; component) \times P(reliability \mid components \; that \; affect \; reliability) \times P(restoration \mid reliability, \; components \; that \; affect \; restoration) \times P(resilience = True \mid reliability, \; restoration) \tag{4}$$

To sum up, we need to analyze the dependency relationship according to the component diagram, and then build a Bayesian network. After inputting parameters such as conditional probability table, the resilience is calculated through Eq. (4).

4.3. Implementation of dynamic Bayesian network method

To apply the dynamic Bayesian network method to the "4+1" view model, a Bayesian network is constructed by analyzing the *activity diagram* structure. We utilize methods for constructing Bayesian networks based on *circle-serial-parallel*, and *minimal paths* (Li et al., 2016; Locks, 1978). In addition, to combine the activity diagram and architecture style, we calculate the node recovery rate based on the component diagram that corresponds to the activity diagram nodes. By calculating the availability of each time slice, it is possible to create a set of time-varying system availability and then calculate the system's resilience using the resilience calculation formula.

In our method, all nodes are attacked with a maximum of n types of attacks. In the second time slice, all nodes are attacked by i ($1 \leq i \leq n$) types of attacks. Therefore, the prior probability of node failure after attacks is $P_i = \frac{i}{n+1}$.

Due to the restoration of the system, a damaged node may be restored to its normal state quickly. This method simplifies the recovery rate of nodes under different attacks by defining it as $\mu_i = (1 - P_i)\mu$, where μ is the recovery rate under normal conditions.

In addition, the state of all activity diagram nodes will change over time. Based on the Markov assumption, the probability of node transition between time slices t and $t + \Delta t$ is as (5) to (8):

$$p(X_{t+\Delta t} = word|X_t = work) = e^{-\lambda\Delta t} \tag{5}$$

$$p(X_{t+\Delta t} = fail|X_t = work) = 1 - e^{-\lambda\Delta t} \tag{6}$$

$$p(X_{t+\Delta t} = fail|X_t = fail) = e^{-\mu\Delta t} \tag{7}$$

$$p(X_{t+\Delta t} = word|X_t = fail) = 1 - e^{-\mu\Delta t} \tag{8}$$

Where, λ and μ represent the failure rate and recovery rate of the node respectively.

Since the resilience value increases with availability A and decreases with recovery time t, the value of resilience is represented by A/ln(t). Therefore, the formula for calculating resilience is (9).

$$\rho = \frac{A}{nln(t_1)} \sum_{i=1}^{n} \frac{A_2^i A_3^i}{ln(t_3^i - t_2^i)} \tag{9}$$

Where, n is the maximum number of external attacks, and A is the steady-state availability before the attack. A_2^i and A_3^i are the post-attack transient availability and the new steady-state availability respectively when the number of attack types is i. t_1 is the time when the steady-state availability A is reached. When the number of attack types is i, t_2^i is the moment of being attacked, and t_3^i is the moment of reaching a new steady-state. Notably, there is no true steady state availability. Therefore, in this method, steady-state availability is defined as the availability when the difference of 5 consecutive time slices is equal to or less than 10^{-5} (Cai et al., 2018).

Generally speaking, we need to analyze activity diagrams and build two Bayesian networks based on *circle serial parallel*, and *minimal paths*. After inputting the parameters such as conditional probability table, failure rate and recovery rate, the simulation is carried out to obtain the system availability that changes with time. Finally, the resilience value is calculated by Eq. (9).

4.4. Implementation of minimal path method

We compute the minimal path by analyzing the structure of the *activity diagram*. The minimal path calculation method we use is the minimum path tree algorithm (Jasmon and Kai, 1985). And this method uses three metrics of robustness, rapidity and availability to comprehensively evaluate the resilience (Zhu et al., 2019). Besides, in order to combine the activity diagram with the architectural style, we calculate the recovery rate of the nodes according to the component diagram components corresponding to the activity diagram nodes.

The path state change function describes the change of the path over time. In the process of simulation, there are two states S of each node. Similarly, there are also two states P of each path. Therefore, for a path P with n nodes, the path state calculation formula at time t is (10).

$$P(t) = \prod_{k=1}^{n} S_k(t) \tag{10}$$

where $S_k(t)$ is the state of the kth node on the path at time t.

Then, for a system with M paths, the calculation formula of its performance function Q(t) is as (11):

$$Q(t) = \sum_{m=1}^{M} w_m \cdot P_m(t) \cdot 100\% \tag{11}$$

where, w_m is the weight of the mth path, and p_m is the state of the mth path.

As mentioned earlier, we quantify resilience through three metrics: robustness, rapidity, and availability. Through simulation, we can get the system performance as a function of time. Then the values of the three metrics can be calculated.

The calculation formula of robustness is shown in (12):

$$Robustness_i = min\{\frac{Q_{min_{i,u}}}{100\%}\} \tag{12}$$

where $Robustness_i$ is the robustness evaluation result under the ith simulation, $Q_{min_{i,u}}$ is the valley value of the performance under the uth disturbance of the ith simulation, where $u \in [1, U_i]$, and U_i is the number of times the disturbance occurs under the ith simulation.

Under the uth disturbance in the ith simulation, the calculation formula of the speed is as (13):

$$V_{i,u} = \frac{QL_{i,u}}{RL_{i,u}} = \frac{100\% - Q_{min_{i,u}}}{\sum_{t=t_{begin_{i,u}}}^{t=t_{end_{i,u}}} (100\% - Q_i(t))} \tag{13}$$

where, $QL_{i,u}$ is the maximum performance loss and $RL_{i,u}$ is the total performance loss under the uth disturbance in the ith simulation.

Therefore, the rapidity of the system in the ith simulation can be taken as the average value of the recovery speed under U_i disturbances, and the calculation formula is as (14):

$$Rapidity_i = \frac{\sum_{u=1}^{U_i} V_{i,u}}{U_i} \tag{14}$$

In the ith simulation, the availability of the mth service of the system is the proportion of its normal state time in the simulation period T, and the expression is as (15):

$$Availability_i = \frac{\sum_{t=1}^{T} P_m(t)}{T} \tag{15}$$

Therefore, the resilience calculation formula is shown in (16).

$$Resilience = \alpha_1 \cdot Robustness + \alpha_2 \cdot Rapidity + \alpha_3 \cdot Availability \quad (16)$$

Where, α is the weight of different metrics.

The weight assignment method we use is the independent weight coefficient, which uses the collinearity between the metrics to determine the weight (Bruneau et al., 2003). The calculation formula of each weight is shown in (17):

$$\alpha_j = \frac{\frac{1}{R_j}}{\sum_{j=1}^{J} \frac{1}{R_j}} \quad (17)$$

Where, R_j is the complex correlation coefficient of the jth metric (Bruneau et al., 2003).

In general, the method first analyzes the activity diagram to obtain all minimal paths, and regards each path as a business from the source point to the sink point. Then use the minimal path to model the system performance, and then use the performance function to quantify each metric. Finally, metrics is used to comprehensively evaluate the system resilience.

4.5. Implementation of Markov Modeling method

Markov modeling method mainly focuses on the reliability and restoration of software system architectures. Firstly we assess the reliability of the activity diagram nodes based on the reliability of the components and the component relationships between the components in component diagram, then we construct a Markov model based on the activity diagram, and calculate the state transfer matrix of the Markov model based on the reliability of the activity diagram nodes. Finally, we use the state transition relationships of the Markov model to simulate the possible state transitions of the system during actual operation and assess the resilience of the system based on these state transitions.

More formally, a Markov model of a software architecture contains two parts: the states $s_1, s_2, s_3, \ldots, s_k, S, F$ and the transition matrix T. States s_1 to s_k are normal states directly generated from the activity diagram, and S and F are special states that denote the success and failure states of the architecture model. The matrix T is constructed according to the relationships of activity nodes and component relationships.

The architecture resilience is then calculated by (18):

$$resilience = (-1)^{k+1} R_k \frac{|E|}{|I - M|} \quad (18)$$

In (18), R_k is the resilience of last state node, M is the submatrix of T except the first 2 columns and 2 rows, I is the identity matrix with the same shape of M, E is the adjoint matrix of M at $[1, k]$.

4.6. The comparison of above five methods

Although all five methods in this paper have been applied to architecture resilience evaluation, they actually focus on different contexts. Depending on the modeling approach, the different methods focus on different sub-attributes of resilience, and Fig. 1 depicts the relationship between the six sub-attributes and the five methods.

Additionally, although all five methods in this paper use the "4+1" view model as input, the focus of each method is also different, so this means that not all methods require the activity and component diagrams as input, the difference of method inputs are shown in Table 2.

As it is often necessary to evaluate large-scale software systems in practical applications, the complexity of the algorithm is related to the cost and resource requirements of the resilience evaluation, especially before evaluating very large and complex systems, it is important to consider the time and computational resources required by the evaluation methods. Table 3 compares the time complexity and space complexity of the five resilience evaluation methods. According to the table, we can find that: the dynamic Bayesian method requires the most resources of the five methods and may be more suitable for resilience evaluation of smaller scale systems; while the attack surface method and Markov method require fewer resources and so may be more efficient for application on large scale systems.

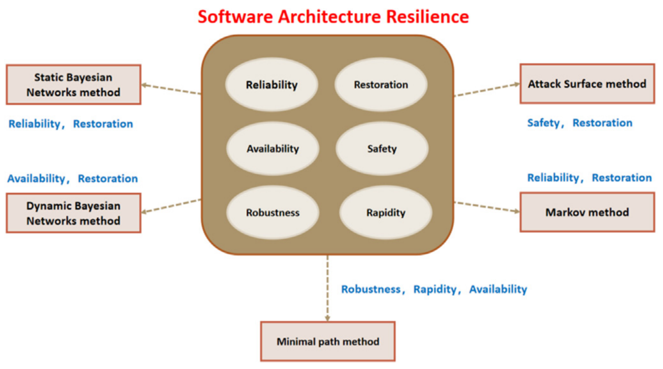

Fig. 1. Each method's resilience context.

Table 2
Input of five methods.

Evaluation method	Activity diagram	Component diagram
Markov modeling	✓	✓
Dynamic Bayes	✓	✓
Static Bayes		✓
Attack surface	✓	✓
Minimal path	✓	✓

Table 3
Complexity of different methods.

Evaluation method	Time complexity	Space complexity
Attack surface	O(M)[a]	O(M)
Markov modeling	O(M^3)	O(M^2)
Dynamic Bayes	O(2^M)	O(2^M)
Minimal path	O(2^M)[b]	O(2^M)
Static Bayes	O(N)	O(N)

[a] M refers to the number of nodes inside input activity diagram.
[b] N refers to the number of nodes inside input component diagram.

5. Experiments and analysis

In this section, we will try to find the answers to following four research questions by comprehensive experiments, analyzes and comparisons etc. Experiments 1–3 will be conducted based on software architecture cases with different architectural styles and their evaluation versions.

The four research questions we concerned are as follows.

- *RQ1*: Are all five methods effective in evaluating software architecture resilience?
- *RQ2*: How architecture resilience changes during the life cycle of software?
- *RQ3*: How architecture resilience influenced by software architecture patterns/styles?
- *RQ4*: Which other factors of software system influence architecture resilience?

All the input data for experiments in this section are designed and modeled by PlantUML tool, there are six experiment object systems in total, which are online shopping system, stock management system, Song Ordering system, Schedule Inquiry system, Train Ticketing system and KWIC system. For example, we use PlantUML to draw the activity diagram and component diagram of these systems as the input of our evaluation methods.

5.1. *Experiment 1: Activity diagram changes*

5.1.1. *Experiment purpose*

The objective of this experiment is to explore how changes in the process view of software architecture influence software resilience evaluated by our five resilience evaluation methods. By evaluating and comparing the resilience of several different versions of a stock management system using five methods, we attempt to find the consistency in those five methods and the important factors affecting resilience in the process view of "4+1" architecture model.

5.1.2. *Experiment objects*

The stock management system is a software system used by the management of a company to store, manage and update various data and information related to warehousing, including customer information, billing data, stock information, and product data. The main part of the system, as well as the database, is stored on a server somewhere within the company. Administrators can log into the system via a client program on their device and initiate requests to the server for queries or changes to data, and the server responds to these requests based on the content of the user's request and the data in the database.

The basic process of using this system is shown in Fig. 1 as an activity diagram: firstly, the user need to send a login request to the server, and send the user name and password after receiving a login permission from server, after checking the validity of user information, the server will send a successful login result to client program and then client program can access data from the database and conduct operations. The first version of this system is called stock management system v1.

An alternative login process is that instead of sending a login request before sending the user Id and password, the client just sends the login request with user Id and password in the login request to the server, this version of system is called stock management system v2.

During the development and usage of this system, managers find that there are also needs of managing the quality of stocks, so a new function is added to the system as managing the quality information: the user is also able to access and manage the quality data after successfully log into the system, in the activity diagram, that means we shall add a path to a new node "manage quality" after node "access internal functionalities". This is stock management system v3.

After the system had been in use for some time, the increasing data volume and complexity made it difficult for a small number of stock managers to maintain the complex system, and management found that it was more efficient to have different administrations managing the various sub-sections of the stock management system than the previous approach. However, the stock management system also needed to be upgraded to support different user permissions: for example, administrators in the customer management section only had access to and could modify customer information, but not view billing data or modify product quality information. So the software developers added a step to check user permissions and set user rights before allowing them to operate the system. This is stock management system v4.

Additionally, for the management system version1, we added a model of the system without loops to investigate the effect of loops on system resilience by assessing and comparing the resilience of the system before and after removing the loops. This is stock management system v5.

Table 4
Resilience of different version of stock management system.

Evaluation method	System version				
	1	2	3	4	5
Markov modeling	0.248	0.31	0.313	0.310	0.554
Attack surface	0.642	0.693	0.709	0.519	0.571
Static Bayes	0.834	0.834	0.834	0.834	0.834
Dynamic Bayes(1)[a]	0.400	0.401	0.406	0.382	0.383
Dynamic Bayes(2)[b]	0.403	0.406	0.411	0.383	0.397
Minimal path	0.628	0.644	0.646	0.606	0.610

[a]Implementation of Dynamic Bayes method using circle-serial-parallel.
[b]Implementation of Dynamic Bayes method using minimal path.

5.1.3. *Experiment results and analysis*

Following the evaluation step listed in the implementation section, we perform resilience evaluation on all five versions of stock management system with five methods, the results are shown in Table 4.

From the Table 4, we can note that the individual methods differ in the values assessed for the five methods of resilience, but in general, there is some consistency in the trend of the different versions of resilience results for the four methods, except for the static Bayesian method. All four methods agree that version 2 has better system resilience than version 1, while version 3 has better resilience than version 2 and version 4 has less resilience compared to version 3. The main difference between version 5 and version 4 is that version 5 removes the rings from the activity diagram, whereas for the minimal path analysis-based minimal path algorithm and the path-based dynamic Bayesian approach, the rings in the activity diagram are ignored in the calculation of the path, so there is no significant change in resilience when the rings are removed.

To explore the factors in the architecture that affect resilience, we need to further analyze the similarities and differences between the different versions of the architecture. Firstly, we compare the differences between version 1 and version 2: the process of version 2 has less steps than that of version 1: the client no longer needs to send a login request, and the server no longer needs to return a permission to login request; instead, the client directly sends the username and password directly to the server. This reduction in flow actually reduces the risk of the system receiving an attack, or rather, makes it easier for the system to recover. For example, for dynamic Bayes method, when using path to construct model, this means a shorter average path length in the model, and the same for minimal path method. Additionally, from the point of view of the attack surface, version 2 has smaller attack surface: the log in request and log in permission can both be untrusted data items, which can be used by some attackers.

We found by comparison between versions 1, 2, 3 and 4 that reducing the serial nodes in the process view of the system, or reducing the mean (minimal) path length, improves the resilience of the system, and conversely, if the mean (minimal path) length of the activity diagram in the system increases, the resilience of the system generally becomes worse. Alternatively, adding feasible paths to the process view of the system (or adding business processes, adding parallel nodes) can improve the resilience of the system. About version 5, also the problem of adding and removing circles in the process view, sadly, static Bayes, dynamic Bayes with path and minimal path method are not sensitive to circles in the process view, but the remaining methods do believe that models without cycles are more resilient.

Table 5
Resilience of different version of online shopping system.

Evaluation method	System version			
	1	2	3	4
Markov modeling	0.533	0.749	0.783	0.891
Attack surface	0.472	0.523	0.524	0.528
Static Bayes	0.787	0.795	0.801	0.808
Dynamic Bayes(1)[a]	0.350	0.378	0.385	0.393
Dynamic Bayes(2)[b]	0.380	0.385	0.405	0.413
Minimal path	0.593	0.614	0.623	0.612

[a]Implementation of Dynamic Bayes method using circle-serial-parallel.
[b]Implementation of Dynamic Bayes method using minimal path.

5.2. Experiment 2: Component diagram changes

5.2.1. Experiment purpose

The objective of this experiment is to explore the impact of changes in the components and component relationships in the software architecture model on the results of the resilience evaluation. Several versions of architecture models of an online shopping system at different stages of evolution are used to accomplish this task.

5.2.2. Experiment objects

At first, the online shopping system was just a website for some small-scale merchant, so it was designed as a simple three-tier structure: the customer accesses the website of the online shop via a browser, the server has access to certain data in the database, and there is only a single server and a single database for all the services of this online shopping system. We call this the online shopping system v1. However, soon afterwards, the huge increase in data made it difficult to maintain and update the single database, and the single database was not conducive to storing data securely. The merchant decided to use three different databases to store different types of information: a user database to store user-related data, a product information database to store product information and storage, and an order database to store order information and logistics information. This is the online shopping system v2. However, the company's online shopping business was growing rapidly with the increasing popularity of the Internet in the region. The massive increase in the number of users and access demand was putting a heavy burden on the servers, and the increase in response time and reduced response efficiency was making the user experience poor, so the company decided to transform the online shopping system into a microservice architecture: each type of sub-service was handled by a separate server to reduce the load on each server, while also making the architecture simpler and easier to maintain. This is online shopping system v3. And finally, for the purpose of data security and make the system better recover from disruptions, backup databases are used to save additionally copy of each kind of data inside the system, and that is online shopping system v4.

5.2.3. Experiment results and analysis

The experiment process of this experiment is similar to experiment 1, the results are shown in Table 5.

According to the resilience evaluation results in Table 5, we can find that most of our five methods are consistent when assessing the changes between versions, except for the minimal path method, it shows the inconsistency for online shopping system v4 from other four methods, the resilience is decreased by about 0.01 compared to the previous version while all other four methods show that the resilience is increased.

Let us see what happened, when the online shopping system evolves from v1 to v2, a distributed database was used in v2 instead of a single database used in v1, all five methods show that this change improves the resilience of the architecture, and for all five evaluation methods except the dynamic Bayesian method, and we find that this change increases the value of resilience most obviously among all three changes performed. The evolution from v1 to v2 increases the architectural resilience of the system, by reducing the coupling and increasing the reliability and robustness of the system, showing that the resilience of the architecture is related to the robustness and reliability of the system.

As for the changes from v2 to v3, when the microservice architecture was applied to the system, although the initial purpose of this change was only to improve the system's ability to cope with large scale accesses (increasing the availability and reliability of the system), this change also improved the resilience of the system, all methods evaluate the v3 version with higher resilience values than v2 and v1, which demonstrates an intrinsic link between architectural resilience and the availability and reliability of the system.

The last change to the online shopping system was aimed at improving the safety and restoration of the system, which was achieved by adding additional backup databases and security components to the system, all methods but the minimal path method found this change can improve the architectural resilience of the system (which means increasing the safety and restoration of the system is beneficial in improving the resilience). For the minimal path method, which considers the v4 version to be less resilient than the previous version, this is because of a combination of two reasons: compared to previous versions, the changes in the online shopping system v4 mainly focus on improving safety and recovery, which the minimal path method is not sensitive to, so the resilience evaluated by the minimal path method did not show an improvement like other methods; and because v4 adds additional components to maintain security and data recovery, the system will need more time to recover from a disruption (more time to maintain these components), so the system rapidity will decrease for this version, thus affecting the final resilience value.

In summary, this part of the experiment explores the impact of changes in the composition and pattern of system components on architecture resilience, and by analyzing the results we find that: system resilience is indeed closely related to its sub-attributes; and the principle of low-coupling, high-cohesion architectural construction has a significant impact on architecture resilience.

5.3. Experiment 3: Architecture style changes

5.3.1. Experiment purpose

The purpose of this experiment is to investigate the effect of various software architectural styles on the evaluation of resilience using the five resilience evaluation methods. By comparing the results of the resilience evaluation, we try to identify which architecture style has better resilience and find the consistency of the evaluation results of the five methods.

5.3.2. Experiment objects

This experiment takes the development view and process view of KTV song ordering system, schedule inquiry system, online shopping system, train ticketing system, KWIC system and stock management system realized by different architectural styles as experimental objects. Table 6 shows the correspondence between six systems and five architectural styles.

The connection mode of some components distinguishes the component diagram implemented in B/S style (BS), C/S style (CS) and Pipes/Filters style (PF). There are two connection modes: direct connection and interface connection. The Main program/Sub program style (MS) is a call/return structure that separates the

Table 6
Correspondence between system and architecture style.

Experiment subject	Architecture style				
	BS	CS	PF	MS	OO
Song ordering system	✓	✓	✓	✓	✓
Schedule inquiry system	✓	✓	✓	✓	
Online shopping system	✓	✓	✓		✓
Train ticketing system	✓	✓			
KWIC system	✓	✓	✓	✓	✓
Stock management system	✓	✓	✓		✓

problem into multiple processing phases. The main program's correctness depends on the accuracy of the sub-programs it invokes. Object Oriented style (OO) separates the software into discrete, reusable objects, each containing data and behavior linked to the object.

5.3.3. Experiment results and analysis

Based on the process of five resilience evaluation methods, we have conducted resilience evaluation for the above six systems. The evaluation results are showed in Table 7.

By comparing the results, we find that the architectural style has a significant effect on the resilience. In general, the architectures implemented through the Object Oriented style have the highest resilience value. B/S style has higher resilience value than C/S style. The resilience of Pipes/Filters style and Main program/Sub program style is poor.

Combining the component diagram structures and evaluation results of different architectural styles, we can draw the following conclusions:

- Interfaces can reduce the architecture's coupling degree, thereby effectively improving the resilience. For the B/S style, the components are connected through interfaces, and the coupling degree is low, so the resilience is better. The Pipes/Filters style is less resilient due to the high degree of coupling between components. The Main program/Sub program style also has poor resilience due to the strong dependency between the main program and the sub program.
- For the Static Bayesian network method, the use of interfaces can reduce the dependence on other components. When the parent node is difficult to operate normally, the child node still has a high probability to operate normally, thus improving the system resilience.
- For the Dynamic Bayesian Network and Minimal Path method, reducing the coupling between components can improve the probability of normal operation of components, thereby improving the recovery rate of activity diagram nodes related to these components, thus improving the resilience.
- For Attack Surface method, a strong dependency will make it difficult for the system to recover from the attack. Therefore, using interfaces can reduce coupling and improve resilience.
- For Markov Modeling method, reducing the coupling degree of the system can make the damage area of the system smaller when attacked, and also make the system easier to recover from the attack.

By comparing the evaluation results of different systems under the same architecture style, we believe that:

- The size of the attack surface will affect the resilience of the system. The larger the attack surface, the worse the resilience.
- A more complex activity diagram means that the system business process is more complex, resulting in a lower resilience value. This is because a complex process means more potential attack points and more difficult to recover from attacks.
- When there are components that undertake multiple businesses, the system's resilience value will be low. This is because damage to these components will result in damage to multiple functions of the system.

5.4. Answer to research questions

Table 8 shows the corresponding relationship between research questions and experiments. Based on the results and analysis of the above experiments, we may provide the following answers to the four research questions.

Answer to RQ1: We find that all five resilience evaluation methods studied in this paper are effective. Although they differ in resilience values, they are consistent in the trend of resilience change and design styles. It should be noted that the static Bayesian method performs differently than other evaluation methods in certain circumstances since it does not take activity diagram into consideration.

Answer to RQ2: We find that changes in architectural resilience are actually related to specific properties and component relationships in the architecture: in the process view, fewer serial nodes or fewer loops typically imply better resilience, whereas more selective nodes make the system more resilient. In terms of the component interactions of a system, a lower degree of interdependence between components might result in greater resilience.

Answer to RQ3: We find that systems designed in the object oriented style are generally more resilient than the majority of other design styles studied. Systems designed in the B/S style are also generally more resilient, especially compared to those designed in the C/S or pipe filter style. Finally, systems designed in the main/sub program style are generally less resilient.

Answer to RQ4: We find that architectural resilience is related to a number of intrinsic architectural properties and characteristics, like attack surface and minimal path. In terms of the attack surface of a system, systems with a smaller attack surface tend to be more resilient. In terms of the minimal paths present in a system, systems with a shorter average minimal path length are generally more resilient. Reducing direct dependencies between components and increasing the usage of APIs and interfaces will increase the system's resilience. Reducing coupling will also contribute to the system's resilience; architectures that adhere to the open-closed concept and the low coupling, high cohesion principle are usually more resilient.

In addition, we can find that the scalability of all the five methods are good, it is easy to use them to evaluate the architecture resilience of larger system without changing the method itself. However, the minimal path is the best one of them in the aspect of scalability because of its low cost for computing minimal paths.

6. Conclusion and future work

In this paper, we have done a widespread empirical study for evaluating five representative software architecture resilience evaluation methods, and provided some valuable information to answer our research questions, which tell us which methods are better than others in which aspects.

Of course, there are still two limitations with this study: ① due to the complexity of contemporary software, big, distributed, and sophisticated systems are frequently the focus of architecture resilience evaluations in real-world applications. However, the experiments in this study are based on just a few very modest software architecture models, making it difficult to experimentally show the applicability of the methods and conclusions to

Table 7
Different system resilience evaluation results implemented by different architectural styles.

Experiment subject	Architecture style	Evaluation method					
		Static Bayes	Dynamic Bayes (1)[a]	Dynamic Bayes (2)[b]	Attack surface	Markov modeling	Minimal path
Song Ordering System	BS	0.736	0.486	0.430	0.818	0.937	0.666
	CS	0.718	0.484	0.424	0.810	0.928	0.661
	PF	0.675	0.475	0.414	0.798	0.895	0.637
	MS	0.708	0.453	0.384	0.798	0.921	0.665
	OO	0.740	0.501	0.431	0.805	0.942	0.693
Schedule Inquiry System	BS	0.726	0.510	0.421	0.419	0.904	0.650
	CS	0.711	0.492	0.403	0.413	0.838	0.624
	PF	0.675	0.475	0.392	0.410	0.800	0.611
	MS	0.691	0.478	0.395	0.406	0.737	0.617
Online Shopping System	BS	0.760	0.437	0.329	0.584	0.836	0.625
	CS	0.752	0.418	0.320	0.583	0.804	0.622
	PF	0.716	0.412	0.300	0.553	0.776	0.611
	OO	0.793	0.458	0.340	0.611	0.889	0.648
Train Ticketing System	BS	0.640	0.419	0.316	0.587	0.812	0.583
	CS	0.622	0.305	0.244	0.572	0.717	0.509
KWIC System	BS	0.654	0.450	0.451	0.787	0.932	0.658
	CS	0.587	0.438	0.445	0.776	0.932	0.624
	PF	0.510	0.426	0.415	0.766	0.888	0.593
	MS	0.456	0.401	0.393	0.709	0.894	0.544
	OO	0.832	0.488	0.471	0.806	0.960	0.768
Stock Management System	BS	0.840	0.424	0.325	0.863	0.860	0.678
	CS	0.834	0.418	0.316	0.833	0.452	0.662
	PF	0.733	0.401	0.311	0.838	0.443	0.621
	OO	0.840	0.428	0.322	0.870	0.861	0.662

[a]The implementation of Dynamic Bayes method using circle-serial-parallel.
[b]The implementation of Dynamic Bayes method using minimal path.

Table 8
Corresponding relationship between research questions and experiments.

RQ	Experiment		
	Experiment1	Experiment2	Experiment3
RQ1	✓	✓	✓
RQ2	✓	✓	
RQ3			✓
RQ4	✓	✓	✓

large-scale systems. ② based on the activity diagram and component diagram, we solely examine two views in architecture model when choosing the inputs for the tests. In practice, however, the environment in which the components are deployed, the status of the system's maintenance workers, and even the individual application scenarios will impact the system's attributes and therefore its resilience. If other views from the "4+1" view model can be included in the experimentation process, the conclusions of this study will be more believable and simpler to translate to real-world applications.

Consequently, we think that future research could concentrate on two main directions: ① investigating the effectiveness of these resilience evaluation methods on larger-scale real systems; ② considering the use of multidimensional and more views in architecture models to achieve more realistic resilience evaluation results.

CRediT authorship contribution statement

Jiaxin Pan: The empirical study of both Attack Surface method and Markov Modeling method. **Zixuan Liu:** The empirical study of both Static Bayesian Network method and Dynamic Bayesian Network method. **Donglin Li:** The empirical study of Minimal Path method. **Lulu Wang:** The main idea of the empirical study. **Bixin Li:** Supervisor of Jiaxin Pan, Zhixuan Liu and Donglin Li, and initiate and guide this research.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This work is supported in part by National Natural Science Foundation of China under Grant 61872078 and Huawei Technologies Co., Ltd, China under Grand TC20210817028. Many thanks to Hao Chen, Shengyang Zhou, Chen Chen, and Zheng Dai for their valuable suggestions to early version of this paper.

References

Ahmed, A., Hussain, S.S., 2007. Meta-model of resilient information system.

Bishop, M., Carvalho, M., Ford, R., Mayron, L.M., 2011. Resilience is more than availability. In: Proceedings of the 2011 New Security Paradigms Workshop. pp. 95–104.

Bruneau, M., Chang, S.E., Eguchi, R.T., Lee, G.C., O'Rourke, T.D., Reinhorn, A.M., Shinozuka, M., Tierney, K., Wallace, W.A., Von Winterfeldt, D., 2003. A framework to quantitatively assess and enhance the seismic resilience of communities. Earthq. Spectra 19 (4), 733–752.

Cai, B., Xie, M., Liu, Y., Liu, Y., Feng, Q., 2018. Availability-based engineering resilience metric and its corresponding evaluation methodology. Reliab. Eng. Syst. Saf. 172, 216–224.

Cámara, J., Correia, P., de Lemos, R., Vieira, M., 2014. Empirical resilience evaluation of an architecture-based self-adaptive software system. In: Proceedings of the 10th International ACM Sigsoft Conference on Quality of Software Architectures. QoSA '14, Association for Computing Machinery, New York, NY, USA, pp. 63–72.

Cámara, J., Lemos, R., Vieira, M., Almeida, R., Ventura, R., 2013. Architecture-based resilience evaluation for self-adaptive systems. Computing.

Cimellaro, G.P., Reinhorn, A.M., Bruneau, M., 2010. Framework for analytical quantification of disaster resilience. Eng. Struct. 32 (11), 3639–3649.

Ferris, T.L., 2019. A resilience measure to guide system design and management. IEEE Syst. J. 13 (4), 3708–3715.

Ford, R., Carvalho, M., Mayron, L., Bishop, M., 2013. Antimalware software: Do we measure resilience? In: 2013 Workshop on Anti-Malware Testing Research. IEEE, pp. 1–7.

Gagniuc, P., 2017. Markov Chains: From Theory to Implementation and Experimentation.

Gama Dessavre, D., Ramirez-Marquez, J.E., Barker, K., 2016. Multidimensional approach to complex system resilience analysis. Reliab. Eng. Syst. Saf. 149, 34–43.

Holling, C.S., 1973. Resilience and stability of ecological systems. Annu. Rev. Ecol. Syst. 1–23.

Hosseini, S., Yodo, N., Wang, P., 2014. Resilience modeling and quantification for design of complex engineered systems using bayesian networks. In: Proceedings of the ASME Design Engineering Technical Conference. Vol. 2.

Hutchison, D., Sterbenz, J., 2009. ResiliNets: resilient and survivable networks. ERCIM News 2009.

Jasmon, G., Kai, O., 1985. A new technique in minimal path and cutset evaluation. IEEE Trans. Reliab. 34 (2), 136–143.

Kruchten, P., 1995. Architectural blueprints—the "4+ 1" view model of software architecture. IEEE Softw. 12 (6).

Li, Z., Li, X., Kang, R., 2016. System resilience modeling and assessment based on minimal paths. In: 2016 Prognostics and System Health Management Conference (PHM-Chengdu). pp. 1–4.

Liu, D., Deters, R., Zhang, W.-J., 2010. Architectural design for resilience. Enterp. Inf. Syst. 4 (2), 137–152.

Liu, Y., Li, X.-Y., Lin, Y., Kang, R., Xiao, L., 2017. A colored generalized stochastic Petri net simulation model for service reliability evaluation of active-active cloud data center based on IT infrastructure. pp. 51–56.

Liu, Y., Li, X., Xiao, L., 2018. Service oriented resilience strategy for cloud data center. In: 2018 IEEE International Conference on Software Quality, Reliability and Security Companion. QRS-C, IEEE, pp. 269–274.

Locks, M.O., 1978. Relationship between minimal path sets and cut sets. IEEE Trans. Reliab. 27 (2), 106–152.

Manadhata, P.K., Wing, J.M., 2011. An attack surface metric. IEEE Trans. Softw. Eng. 37 (3), 371–386.

Maza, S., Megouas, O., 2021. Framework for trustworthiness in software development. Int. J. Perform. Eng. 17 (2), 241–252.

Ouchani, S., Lenzini, G., 2014. Attacks generation by detecting attack surfaces. Procedia Comput. Sci. 32, 529–536, The 5th International Conference on Ambient Systems, Networks and Technologies (ANT-2014), the 4th International Conference on Sustainable Energy Information Technology (SEIT-2014).

Proag, V., 2014. Assessing and measuring resilience. Procedia Econ. Finance 18, 222–229, 4th International Conference on Building Resilience, Incorporating the 3rd Annual Conference of the ANDROID Disaster Resilience Network, 8th – 11th September 2014, Salford Quays, United Kingdom.

Sanjay, R., Gupta, H., Seth, A., 2022. A multi-layer feed forward network intrusion detection system using individual component optimization methodology for cloud computing. Int. J. Performab. Eng. 18 (11), 781–790.

Smith, P., Hutchison, D., Sterbenz, J.P., Schöller, M., Fessi, A., Karaliopoulos, M., Lac, C., Plattner, B., 2011. Network resilience: a systematic approach. IEEE Commun. Mag. 49 (7), 88–97.

Theisen, C., Munaiah, N., Al-Zyoud, M., Carver, J.C., Meneely, A., Williams, L., 2018. Attack surface definitions: A systematic literature review. Inf. Softw. Technol. 104, 94–103.

Wang, W.-L., Pan, D., Chen, M.-H., 2006. Architecture-based software reliability modeling. J. Syst. Softw. 79 (1), 132–146.

Zhang, M., Wang, L., Jajodia, S., Singhal, A., 2021. Network attack surface: lifting the concept of attack surface to the network level for evaluating networks' resilience against zero-day attacks. IEEE Trans. Dependable Secure Comput. 18 (1), 310–324.

Zhu, S., Guo, L., Cui, Y., Xiao, R., Yu, Z., Jin, Y., Fu, R., Zhang, J., Xu, T., Chen, J., et al., 2019. Quality suitability modeling of volatile oil in Chinese Materia Medica–Based on maximum entropy and independent weight coefficient method: Case studies of Atractylodes lancea, Angelica sinensis, Curcuma longa and Atractylodes macrocephala. Ind. Crop. Prod. 142, 111807.

Jiaxin Pan is a master student in The College of Software Engineering Southeast University, his research interesting includes software architecture safety and security etc.

Zixuan Liu is a master student in The College of Software Engineering Southeast University, his research interesting includes software architecture safety and security etc.

Donglin Li is a master student in The College of Software Engineering Southeast University, his research interesting includes software architecture safety and security etc.

Lulu Wang is an associate professor in The College of Software Engineering Southeast University, his research interesting includes software architecture safety and security, software testing, software quality assurance etc.

Bixin Li is a full professor in The College of Software Engineering Southeast University, his research interesting includes software architecture safety and security, software testing, software quality assurance etc.