A Markov Error Propagation Model for Component-based Software Systems

Zijing Tian, Yichen Wang,* and Pengyang Zong

*The University of Texas at Dallas, Richardson, 75080, United States

Science and Technology on Reliability and Environment Engineering Laboratory, Beihang University, Beijing, 100191, China

Abstract

In this paper, we propose a Markov chain-based error propagation model to analyze the reliability of component-based software systems and take measures to make the critical components safer. Because it is difficult to test the whole component-based system, we apply an error propagation model to evaluate the reliability of the system with parameters obtained by preliminary data from existing components and integration testing from two connected components. The main parameters required in our Markov model are the error probability for each component, the error tolerance probability, and the error propagation probability for every two connected components. Our model is applied to compute the reliability of the system, find the most suspicious component during debugging, and protect the critical components. Finally, we simulate the process of these three applications using three different systems on MATLAB.

Keywords: error propagation; Markov chain; component-based software system

(Submitted on May 25, 2018; Revised on July 23, 2018; Accepted on August 5, 2018)

© 2018 Totem Publisher, Inc. All rights reserved.

1. Introduction

As software development techniques make more and more progress, composing paradigms to build a software system is becoming increasingly popular in the software engineering field. For instance, component-based software development and COTS-based (commercial off-the-shelf) software development both realize the idea of assembling existing and newly developed components to build a system. However, the quality of software systems are far from satisfactory [14] and different techniques have been proposed [4,5].

For component-based software systems (CBSSs), the assessment of the quality of the system is a significant problem. To evaluate the reliability of the CBSSs and improve the performance of the entire system, it is necessary to conduct a testing process to obtain required parameters. Most component-based systems are quite complex, which make it difficult to run the entire system to get the probabilistic information for reliability assessments. Instead, in this paper, we evaluate the reliability of the systems using only information from primary data in pre-existing components, unit testing for new added components, and integration testing for every two connected components. With many existing, reusable properties of known components in CBSSs, we can assess the reliability of the system expeditiously without accomplishing system-level testing, which has high costs.

Because the error probability of a single component is easy to obtain in CBSSs, the interaction of connected components and the error propagation behavior in the entire system are worth studying. Numerous papers have been devoted to error propagation analysis, and different terms describe close concepts. A fault activity leads to an appearance of an error. The error will lead to an invalid internal system state, and the state may lead to another error and then cause a failure. Failures are defined according to the system boundary. If a failure happens, we can finally realize something is wrong in this system. This process leads us to analyze fault activation, error propagation, and error detection, referred to as error propagation analysis in this paper. Furthermore, these analyses will help evaluate the reliability of systems and prevent

* Corresponding author.

E-mail address: wangyichen@buaa.edu.cn
systems from unanticipated accidents in critical parts. In our approach, the error propagation model can be used in safety, reliability, and fault detection areas.

Most analyses for component-based systems with the purpose of predicting reliability are based on the Markovian assumption. Popstojanova et al. [9] have classified these component-based models as additive models, state-based, and path-based models. Gokhale et al. [8] use Discrete Time Markov Chain to simulate the control graphs in CBSSs. In this article, we comply with the Markovian assumption in component-based system analysis that the next state of the system is only affected by the current state. Morozov et al. [7] propose a Markov model mainly to estimate the error propagation probability from one component to another in a system. Their approach makes it possible to investigate the error propagation features in a system. However, in their model, if there are n components in this system, they will figure out at least $2^n$ states for error propagation analysis, which is exponential. In our approach, for the same system, we only generate $4n-2$ states, which is polynomial time computable. Moreover, we first define the notion of discrete system time Y that describes how many components have been executed (reduplicative components will be counted as another time) in a system time. With instruments in the control module in a system, we can easily get the order of executed components, and with this information, we can build our Markov model to estimate the error propagation probability from one component to another, find the most suspicious component causing a failure, and protect the critical part of a system.

Paper Organization: In Section II, we present the related study of error propagation models. Section III discusses the Markov chain and Bayes’ theorem applying to component-based system reliability and makes a few assumptions. In Section IV, we propose our Markov model and describe the applications of our model in detail. Our simulation results are shown in Section V. The conclusions are presented in Section VI.

2. Related Study

2.1. Reliability Analysis of Component-based Systems

It is well known that, to estimate the reliability of the entire system, we must have a comprehensive understanding and obtain precise information in every single component. Nevertheless, the interaction between different components, the specification of the system, and the critical problems that need to be addressed are considered this assessment.

In [3], Filieri et al. depict a reliability model for architecture level analysis. They propose a reliability model for each single component, which included Reliability (the probability of a component to be correct), Robustness (the probability of a component to tolerate previous errors), Internal Failure (the probability of a component to have an error by itself), and Switching (the probability of a component to produce an outgoing failure). Then, as a result, they use the parameters in this model to conduct the reliability estimation, sensitivity analysis, and optimal configuration for the entire system. In [6], Mohamed and Zulkernine calculate the reliability of CBSSs by failure scattering behavior, fault localization ability, failure propagation lower bound, failure masking and number of ASRs relationship, and failure propagation and number of ASRs relationship. [1] simulates the failure behavior of a CBSSs based on the architecture through two case studies.

2.2. Error Propagation Analysis

Error propagation in component-based systems is worthy of being studied. A fault resulting error in one component is seldom independent of other components in this system. When we want to estimate the reliability of the entire system or just want to know the probability that an error will occur in a single component, we should know the error propagation characteristics of the system.

Popic et al. in [12] apply error propagation models to predict the reliability of component-based systems. Their approach considers the UML diagrams and uses these diagrams to achieve an early reliability prediction. Then, the error propagation model is introduced to the prediction formula. The defects of this research are the following: (1) the calculation for error propagation probability is too complex and time-consuming, and (2) because they combine UML information, which is obtained in the early design stage of the development, and error propagation information, which should be collected during the testing process, their prediction does not seem convincing. Cortellessa and Grassi [2,13] propose error propagation with the assumption that the error propagation rate (which is transmitted by the previous components) in a component is independent to the error rate of itself. Then, they embed their error propagation model within the reliability evaluation model. The main problem we find in this research is that the error propagation rate of a component should not be independent to the previous components connected to it. The interaction of different components should be concerned with information from both. In [7], the authors present an error model aimed at Operating Systems, which is a transient data level error. They pay more attention to the error type and the influence of the errors. Morozov and Janschek propose their
Markov-based error propagation model. Their model denotes every possible state of the entire system that consists of the next component, activated component, sequence of infected components, and sequence of detected components. As we mentioned before, in a n components system, exponential states will be generated. In addition, when considering the cycle in the system, there may be a state explosion. Hence, the application of this research cannot be easily implemented because in most cases the generation of these states cannot be accomplished by machines. Their approach takes advantages of control flow and data flow graphs to generate their DTMC model, which is very helpful for error propagation analysis. In [15,16], the concept of error propagation is also included in the analysis of software quality.

2.3. Error Propagation Analysis Associated with Virus Propagation

When we first started this approach, we decided to apply some principles of virus propagation and worm propagation to this area. In [11], we learn some concepts in computer virus propagation. Unfortunately, their Markov model for virus propagation is based on CTMC (Continuous-Time Markov Chain), while in CBSSs, we apply DTMC to error propagation analysis. Another serious reason that we cannot map these two propagation processes together is because the propagation probability of different machines can be considered the same while for different components in a system, they are not. However, the research from [10] inspire us to estimate the increasing and decreasing trends of a cycle in the system, and in the early design stage, we can build some error-decreasing subsystem to protect the critical components of the system.

3. Reliability Estimation based on Markov Chain and Bayes’ Theorem

In this section, we will introduce how we apply the Markov chain to the error propagation analysis and reliability estimation for component-based systems.

3.1. Markovian Assumption in Component-based Software System Analysis

In [12], the authors make this assumption: the transfer of control among program modules is a Markov process. This assumption implies that the behavior of the next executing module will depend on the current one only and is independent of the past history. If the modules in the system will not change, the transition probabilities have been shown to be quite consistent for the given user environment. Therefore, as we will calculate the reliability of the system, we need to make these probabilities constant.

One of the reliability models discussed an assumption that failures in different components in one system are independent. In the error propagation analysis for reliability estimation, this assumption cannot hold.

Markov Chain Definition:

A stochastic process \( X=\{X_n|n\geq0\} \) on a countable set \( S \) is a Markov chain if, for any \( i, j \in S \) and \( n\geq0 \),

\[
P\{X_{n+1} = j|X_0, \ldots, X_n\} = P(X_{n+1} = j|X_n) \]

\[
P\{X_{n+1} = j|X_n = i\} = p_{ij}
\]

\( p_{ij} \) is the probability that the Markov chain jumps from one state \( i \) to another state \( j \).

When we apply the Markov chain to a model, we need to assure that the sum of transition probabilities of every two states should be:

\[
p_{ij} \geq 0, i, j \geq 0 \text{ and } i, j = 0, 1, \ldots
\]

\[
\sum_{j=0}^{\infty} p_{ij} = 1
\]

The one-step transition matrix should be:

\[
P = \begin{bmatrix}
p_{00} & p_{01} & p_{02} & \cdots & p_{0n} \\
\vdots & \ddots & \ddots & \vdots & \vdots \\
p_{n0} & p_{n1} & p_{n2} & \cdots & p_{nn}
\end{bmatrix}
\]
In error propagation analysis for component-based systems, we need to assign the states to depict the error propagation characteristics.

3.2. Bayes’ Theorem Applying in our Approach

Another important assumption in our research is based on Bayes’ Theorem: when we find an error in the output of a component and we know the component obtained an error by itself this time, we assume the error is caused by the previous components rather than the current one. With this assumption, we can claim that the probability of a component to transmit an error to the next component is independent of the probability of a component to cause an error (because when these two cases happen simultaneously, we count this situation as the former one). This is an assumption based on the debugging processing for the system. In our approach, when the system produces an error, we will examine the very first suspicious component that may cause this error. If there is more than one error source in this system, we should check the suspicious component that is greater.

Under this assumption, we propose an error propagation model for a single component. For each component, we need to obtain four parameters:

- \( pp \): Introduced error propagation probability.
- \( pt \): Introduced error tolerant probability.
- \( po \): Initiative error occurs probability.
- \( pc \): No introduced or initiative error occurs.

\[ pp + pt = 1; \ po + pc = 1 \]

The probability of a component \( C_i \) to have an introduced error is \( pi \), while the probability that no error has been introduced in a component \( C_i \) is \( pn \).

\[ pi + pn = 1 \]

Parameters \( pp \) and \( pt \) are under the condition that the previous error has propagated into this component, while \( po \) and \( pc \) are obtained when there is no error introduced by the previous components.

If the observed sequence of the executed components is \( \{ C_1, C_2, \cdots, C_n \} \), the probability that an error occurs in \( C_i \) and propagates to \( C_j \) is \( P_{ij} \), the probability of \( C_i \) having an initiative error is \( po_i \), the probability of \( C_i \) propagating an introduced error is \( pp_i \), and the probability that \( C_i \) is correct is \( pc \).

\[ P_{ij} = (1 - P_{i1} - P_{2i} - \cdots - P_{(i-1)i}) po_i \prod_{k=i+1}^{j} pp_k \] (1)

This formula is recursive, and we need the previous computed propagation probabilities to obtain the objective \( P \).

3.3. Estimation of the Reliability for Component-based Software Systems

To estimate the reliability of the system, we need to calculate the probability of every single component in this system to be correct.

\[ PCR_i = 1 - PER_i \] (2)

\( PER_i \) is the probability that a component \( C_i \) contains errors.

With the parameters we defined in Section 3.2, we can calculate \( PER_i \) as:

\[ PER_i = \sum_{q=1}^{i-1} P_{qi} + (1 - P_{i1} - P_{2i} - \cdots - P_{(i-1)i}) po_i \] (3)
The correct probability of the entire system to be correct is:

$$PCR_{\text{system}} = \prod_{i=1}^{n} PCR_i$$

The number $n$ is the total number of components in this system.

In the next section, we will present our Markov-based model and then introduce the application of our simplified model.

4. Markov Error Propagation Model for Component-based Software Systems and Applications

In this section, we show our model of error propagation analysis in CBSSs. We make a more comprehensive consideration about how to measure the error propagation properties. In our model, we declare four sub-parameters instead of just the propagation rate and non-propagation rate proposed by some other researches. The states in our Markov model is at most $4^n$. However, in our reliability estimation method, we only need $4n-2$ states. The remainder of this section describes the applications of our model.

4.1. Markov-based Error Propagation Model for CBSSs Reliability Estimation

In component-based systems, to estimate the reliability of the entire system, we need to record the order of the executed components. Therefore, we propose a conception called discrete system time to record how many components have been executed (repetitive execution will be counted for another time): $Y$.

When we need to estimate the system with cycle, to refrain from state explosion, we need to count $Y$ in a system. In most component-based software systems, there is a control module to assign tasks to components. We can instrument this CMOD to obtain $Y$ in this execution. In some other CBSSs without a CMOD, we should instrument each component to record $Y$.

In a CBSSs, it is expensive and time-consuming to run the entire system to estimate the reliability. Nevertheless, it is convenient to get the error probability of a single component that is pre-existing where we can get the parameter $p_o$ and $p_c$. The $pp$ and $pt$ can be obtained in integration testing.

With all these parameters gained, we can build our Markov-based error propagation model.

Firstly, we build an error propagation model to depict the error propagation features. In a single component, we define four states that correspond to the four parameters we mentioned previously:

- $Sp \leftrightarrow pp$: This component transmits the introduced error.
- $St \leftrightarrow pt$: This component absorbs the introduced error.
- $So \leftrightarrow po$: This component initiatively generates an error.
- $Sc \leftrightarrow pc$: This component and the previous component are correct.

With these states defined for every single component, we can build a model to simulate the error propagation process in component-based software systems, presented in Figure 1.

In Figure 1, which is a directed network graph for a system consist of $n$ components, $Sp(i)$, $St(i)$, $So(i)$, and $Sc(i)$ show the states of the $i^{th}$ component in this system. Each arrow in this graph indicates a probability from one state to another. This Bayesian network shows us a way to analyse the error propagation properties during the execution of the system. For the first component $C_1$, there exists no possible introduced error, so we have only obtained $So$ and $Sc$ for it.

Since we have finished the integration testing to get the parameters we need, we can calculate the probability from states in the $i^{th}$ component to states in $(i+1)^{th}$:

$$P[Sp(i) \rightarrow Sp(i+1)] = P[So(i) \rightarrow Sp(i+1)] = pp_{i+1}$$
For every component in this system, the probability of it being incorrect is the probability of the system being in state $S$ at discrete system time $Y$.

For further studies, we should convert this Bayesian network model into DTMC to simplify the computing process of reliability estimation. In this model, if the total number of the components in this system is $n$, we have a total of $4n-2$ states. To standardize the computation of the probability of a state in system execution, we build a DTMC based on this Bayesian network.

Since we have the transition probability of each possible path, we can build this DTMC as we order the states as: $So(1), Sc(1), Sp(2), St(2), So(2), Sc(2), \ldots, Sp(n), St(n), So(n), Sc(n)$. We mark these $4n-2$ states as: $S_1, S_2, \ldots, S_{4n-2}$. The transition matrix from $S_1$ to $S_{4n-2}$ is a upper triangular $(4n-2) \times (4n-2)$ matrix $M_n$:

$$
\begin{bmatrix}
0 & M_{12} & 0 & 0 & \cdots & 0 \\
0 & 0 & M_{23} & 0 & \cdots & 0 \\
0 & 0 & 0 & M_{34} & \cdots & 0 \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & \cdots & 0 & M_{(n-3)n} \\
0 & 0 & 0 & \cdots & 0 & 0 
\end{bmatrix}
$$

$M_{(i+1)}$ is the sub-matrix of the transition matrix that denotes the transition probabilities of states from component $(i-1)$ to component $i$. $M_{12}$ is a $2 \times 4$ matrix:

$$
\begin{bmatrix}
pp_2 & pt_2 & 0 & 0 \\
0 & 0 & po_2 & pc_2 
\end{bmatrix}
$$
The other sub-matrices $M_{(i+1)}$ are $4 \times 4$ matrices:

\[
\begin{bmatrix}
pp_{i+1} & pt_{i+1} & 0 & 0 \\
0 & 0 & po_{i+1} & pc_{i+1} \\
pp_{i+1} & pt_{i+1} & 0 & 0 \\
0 & 0 & po_{i+1} & pc_{i+1}
\end{bmatrix}
\]

We can only start from $So(1)$ and $Sc(1)$, which are the initial states of this Markov Chain. Using the transition matrix $M_n$, it is possible for us to calculate the error probabilities for the $i^{th}$ component.

As we defined before, the error probability of the $i^{th}$ component is the probability of the final state to be $Sp(i)$ or $So(i)$ in this Markov process. Since the start state will be either $So(1)$ or $Sc(1)$ and the discrete system time $T=i$ when the executing component is $i$, the error probability and the correct probability of the $i^{th}$ component can be calculated as:

\[
PER_i = PEP_i + PEO_i \quad (5)
\]

\[
PCR_i = PCT_i + PCC_i \quad (6)
\]

$PEP_i$ describes the probability that the $i^{th}$ component is in state $Sp(i)$, $PEO_i$ describes the probability that the $i^{th}$ component is in state $So(i)$, $PCT_i$ describes the probability that the $i^{th}$ component is in state $St(i)$, and $PCC_i$ describes the probability that the $i^{th}$ component is in state $Sc(i)$.

The transition matrix $M_i$ is the sub-matrix of $M_n$:

\[
\begin{bmatrix}
0 & M_{12} & 0 & 0 & \cdots & 0 \\
0 & 0 & M_{23} & 0 & \cdots & 0 \\
0 & 0 & 0 & M_{34} & \cdots & 0 \\
0 & 0 & 0 & 0 & \cdots & 0 \\
0 & 0 & 0 & \cdots & 0 & M_{(i-1)i} \\
0 & 0 & 0 & \cdots & 0 & 0
\end{bmatrix}
\]

The distribution over states can be written as a stochastic row vertex $x$ computed as:

\[
x^i = x^1(M_i)^{i-1}
\]

\[
x^1 = (po_1, pc_1, 0, 0, \cdots, 0)
\]

\[
x^i = (0, 0, \cdots, 0, PEP_i, PCT_i, PEO_i, PCC_i)
\]

The length of $x^1$ and $x^i$ is the same as the length of $M_i$, which is $4i-2$.

Note that we could just use the $M_n$ as a transition matrix to compute any $i$ in Equation (7). The length of $x^1$ and $x^i$ will be $4n-2$, and the $PEP_i$, $PCT_i$, $PEO_i$, $PCC_i$ will be found in the $(4i-5)^{th}$ to $(4i-2)^{th}$ elements of $x^i$.

The correct probability of the entire system is:

\[
PCR_{\text{system}} = \prod_{i=1}^{n} PCR_i \quad (8)
\]

Note that using the Markov Chain in this case can only make it easier for us to analysis the error propagation property of the CBSSSs. It is much more convenient than Equation (4).
4.2. Fault Localization by our Error Propagation Model

In addition to estimating the reliability of the system, we can also apply our model to fault localization.

By the derivation of the Bayes’ Theorem,

\[ P(A|B) = \frac{P(A \cap B)}{P(B)} \]  

(9)

We can calculate the probability that an error is invoked in the \( i \)th component and transmit to the \( j \)th component in case it does not vanish in the \( j \)th component.

\[ P_{ij} = PCR_{i-1} \cdot PEO_j \prod_{k=i+1}^{j} pp_k \]  

(10)

This is much simpler than Equation (1).

When we figure out the probabilities of errors propagating from every previous component to the \( j \)th component \( P_{1j}, P_{2j}, \ldots, P_{(j-1)j} \), we can find the most likely component that causes the \( j \)th component to contain an error by checking the components with the largest probability to transmit an error to the \( j \)th component.

4.3. Critical Component Protection

As we mentioned before, our model can be applied to CBSSs with cycles in it. During an execution of a cyclic system, we record the discrete system time \( \Upsilon \) and the order of executed components, and then we can use our model to analyze the error propagation property by regarding the repeated components as new ones with the same parameters as the previous ones. We will not suggest an analysis that has many circulations, as too many states would take place in our model and the computation would be too time-consuming.

As an application, we can use our model to estimate the cyclic part of the system and protect the critical component in our system. For example, if we have a cyclic part with four components in a system \( (C_1, C_2, C_3, C_4) \),

![Figure 2. Cyclic Part Example](image)

Assume the probability of this part to have an introduced error from the previous modules is \( a \), and the probability of this part to export an error to the next module is \( b \). We assume the component before \( C_1 \) is \( C_0 \).

In our model, if the cycling time is 0, \( b_0 = PCR_4 \) under the error propagation model with ordered components \( (C_0, C_1, C_2, C_3) \), then the error probability of \( C_0 \) is \( a \). If the cycling time is \( i (i>0) \), \( b_i = PCR_5 \) under the error propagation model with ordered components \( (C_5, C_4, C_3, C_2, C_1) \), note that the first \( C_3 \) denotes the start state of the error propagation model, and the error probability of it should be \( b_{i-1} \).

To estimate if in an additional cycle \( b_i \) increases, we need to calculate the derivation of \( b_i \) with the parameter \( b_{i-1} \), which is marked as \( B \).

\[ \frac{d}{dB} b_i(B) = \frac{d}{dB} PCR_5(B) \]  

(11)

Since \( b_{i-1} \) is a 0-1 probability value, in most cases, we can find if \( b_i \) will increase or decrease by another time of cycle. We define the tolerant circulation to be the circulation that will reduce \( b \) by another time of cycle.
During the design of component-based software systems, when we need to prevent some critical part of the system from unpredictable errors, we should consider building some tolerant circulation right before the critical components.

5. Case Study

In this section, we will apply our model to a component-based software system with simulated parameters on it. The computation is processed in MATLAB. For a simple Make to Order production planning system, we number the components from 1 to 7: \{C_1, C_2, \ldots, C_7\}.

![Figure 2: Architecture of Make to Order production planning [6]](image)

In this system, there are four paths to the end of the system: \{1, 2, 5, 7\}, \{1, 3, 5, 7\}, \{1, 3, 5, 6, 7\}, and \{1, 4, 6, 7\}. For every path, we should calculate the reliability once. For instance, focusing on the third path \{1, 3, 5, 6, 7\}, we give a simulation of reliability estimation for the system.

Reorder these five components in the third path as \{K_1, K_2, K_3, K_4, K_5\}, and the number of states in this system is 18. The initial states of the system are:

\[ x^1 = (0.01, 0.99, 0, 0, \ldots, 0) \]

The Transition Matrix M is \(M_3\), which can be calculated by \(M_{12}, M_{23}, M_{34}, M_{45}\). By Equations (6) and (7), we can obtain the error probability in Table 1:

<table>
<thead>
<tr>
<th>Component</th>
<th>(K_1)</th>
<th>(K_2)</th>
<th>(K_3)</th>
<th>(K_4)</th>
<th>(K_5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reliability</td>
<td>0.9900</td>
<td>0.9742</td>
<td>0.9835</td>
<td>0.9858</td>
<td>0.9613</td>
</tr>
</tbody>
</table>

Then, the reliability of the system for the third path can be calculated by Equation (8):

\[ PCR_{path_3} = \prod_{i=1}^{n} PCR_i = 0.8989 \]

Here, a problem arises: How can we calculate the reliability of the entire system given the reliability of every path? In [7], Andrey Morozov and Klaus Janschek use control flow and data flow graphs to predict the probability of different paths occurring. Nevertheless, in [10], the authors take advantages of UML diagrams to build the reliability model. With the research conducted by previous authors, we achieve the reliability of the whole system through the following:

\[ PCR_{system} = \prod_{i=1}^{n} p(path_i)PCR_{path_i} \quad (12) \]

Where \(p(path_i)\) denotes the probability of a single path occurring and \(n\) is the total number of paths in the system.

6. Conclusions and Future Work

In this paper, we proposed a Markov-based error propagation model to estimate the reliability of component-based software
systems. Similar to previous research on the analysis of component-based software systems, our approach relies on the Markovian assumption and assumes we already have the error data of some components in this system. Unlike others’ research, we declare four states for every single component to better analyse the error propagation properties in a system. Compared with works completed in [3], our model is much simpler and easier to understand. The key problem solved by this paper is estimating the reliability of the entire system without system-level testing, which significantly reduces time and cost for reliability accessing. Our model can also be applied to fault localization during system-level testing. Additionally, with our model, we can take actions to protect crucial parts of a system during the early period of system design.

For future works, we are preparing to combine absorbing states and transient states in our model, which considers only transient states currently. The compatibility of different system structures is also a big challenge for our approach. The parameters for building an error propagation model will spend a large proportion of the Model Establishment cost, and the prediction of probabilities for the distribution of paths requires more reliable ways to obtain instead of just analysing the software specification documents.

References