Analytical Models for Architecture-Based Software Reliability Prediction: A Unification Framework

Swapna S. Gokhale, Member, IEEE, and Kishor S. Trivedi, Fellow, IEEE

Abstract—Traditional approaches to software reliability modeling are black box-based; that is, the software system is considered as a whole, and only its interactions with the outside world are modeled without looking into its internal structure. The black box approach is adequate to characterize the reliability of monolithic, custom, built-to-specification software applications. However, with the widespread use of object-oriented systems design & development, the use of component-based software development is on the rise. Software systems are developed in a heterogeneous (multiple teams in different environments) fashion, and hence it may be inappropriate to model the overall failure process of such systems using one of the several software reliability growth models (black box approach). Predicting the reliability of a software system based on its architecture, and the failure behavior of its components, is thus essential. Most of the research efforts in predicting the reliability of a software system based on its architecture have been focused on developing analytical or state-based models. However, the development of state-based models has been mostly ad hoc with little or no effort devoted towards establishing a unifying framework which compares & contrasts these models. Also, to the best of our knowledge, no attempt has been made to offer an insight into how these models might be applied to real software applications. This paper proposes a unifying framework for state-based models for architecture-based software reliability prediction. The state-based models we consider are the ones in which application architecture is represented either as a discrete time Markov chain (DTMC), or a continuous time Markov chain (CTMC). We illustrate the DTMC-based, and CTMC-based models using examples. A detailed discussion of how the parameters of each model may be estimated, and the life cycle phases when the model may be applied is also provided.

Index Terms—Architecture-based analysis, Markov models, semi-Markov models, software reliability, unification framework.

I. INTRODUCTION

The size and complexity of computer intensive systems has increased in the past decade more rapidly than our ability to design, test, implement, and maintain them. Computer systems are increasingly being used in various active (controlling), and passive (monitoring) applications; and the trend will surely continue in the future. The presence of computers in every appliance, device, service, and activity of our daily lives increases its criticality. Failures have far reaching implications, not all of which are immediately obvious. As the number of users increases, the number of failures from faults also increases. Computer system failures make newspaper headlines because at best they inconvenience people (e.g., malfunctions of home appliances), cause economic damage (e.g., interruptions of banking services), and in the extreme cases cause death (e.g., failures of flight control systems or medical software).

The computer industry has seen uneven progress. With the steadily growing power & reliability of hardware, software reliability has become a major stumbling block in the realization of highly dependable computer systems. The flexibility & integrating potential of software has made more ambitious systems possible. But software technology has failed to keep pace in quality, productivity, cost, and performance relative to hardware.

The impact of software structure on its reliability & correctness has been highlighted as early as 1975–1976 by Parnas [29], and Shooman [31]. Prevalent approaches to software reliability modeling, however, are black box-based, i.e., the software system is considered as a whole, and only its interactions with the outside world are modeled, without looking

ACRONYMS

- DTMC Discrete Time Markov Chain
- CTMC Continuous Time Markov Chain
- SMP Semi-Markov Process
- DAG Directed Acyclic Graph
- SPN Stochastic Petri Net
- MTTA Mean Time to Absorption
- MTTF Mean Time to Failure
- MNTF Mean Number of Steps to Failure
- CFR Constant failure rate
- TDI Time-dependent failure intensity
- pi,j Probability of transition from component i to component j
- pi,j(k) Probability of transition from component i to component j in k steps
- \(\pi_i\) Steady state probability of being in state i
- \(R_i\) Reliability of component i
- \(\lambda_i\) Constant failure rate of component i
- \(\lambda_i(t)\) Time-dependent failure intensity of component i
- \(L_i(t)\) Cumulative time in state i in the interval (0, t)
- \(V_i\) Expected number of visits to state i
- \(R\) Application reliability
- \(a_i\) Expected number of faults that would be detected in infinite time from module i
- \(b_i\) Failure occurrence rate per fault for module i
- \(\mu_i\) Rate of departure from state i for CTMC
- \(T\) Expected time to completion of the application

NOTATION

THE size and complexity of computer intensive systems has increased in the past decade more rapidly than our ability to design, test, implement, and maintain them. Computer systems are increasingly being used in various active (controlling), and passive (monitoring) applications; and the trend will surely continue in the future. The presence of computers in every appliance, device, service, and activity of our daily lives increases its criticality. Failures have far reaching implications, not all of which are immediately obvious. As the number of users increases, the number of failures from faults also increases. Computer system failures make newspaper headlines because at best they inconvenience people (e.g., malfunctions of home appliances), cause economic damage (e.g., interruptions of banking services), and in the extreme cases cause death (e.g., failures of flight control systems or medical software).

The computer industry has seen uneven progress. With the steadily growing power & reliability of hardware, software reliability has become a major stumbling block in the realization of highly dependable computer systems. The flexibility & integrating potential of software has made more ambitious systems possible. But software technology has failed to keep pace in quality, productivity, cost, and performance relative to hardware.

The impact of software structure on its reliability & correctness has been highlighted as early as 1975–1976 by Parnas [29], and Shooman [31]. Prevalent approaches to software reliability modeling, however, are black box-based, i.e., the software system is considered as a whole, and only its interactions with the outside world are modeled, without looking

ACRONYMS

- DTMC Discrete Time Markov Chain
- CTMC Continuous Time Markov Chain
- SMP Semi-Markov Process
- DAG Directed Acyclic Graph
- SPN Stochastic Petri Net
- MTTA Mean Time to Absorption
- MTTF Mean Time to Failure
- MNTF Mean Number of Steps to Failure
- CFR Constant failure rate
- TDI Time-dependent failure intensity
- pi,j Probability of transition from component i to component j
- pi,j(k) Probability of transition from component i to component j in k steps
- \(\pi_i\) Steady state probability of being in state i
- \(R_i\) Reliability of component i
- \(\lambda_i\) Constant failure rate of component i
- \(\lambda_i(t)\) Time-dependent failure intensity of component i
- \(L_i(t)\) Cumulative time in state i in the interval (0, t)
- \(V_i\) Expected number of visits to state i
- \(R\) Application reliability
- \(a_i\) Expected number of faults that would be detected in infinite time from module i
- \(b_i\) Failure occurrence rate per fault for module i
- \(\mu_i\) Rate of departure from state i for CTMC
- \(T\) Expected time to completion of the application

NOTATION

1The singular & plural of an acronym are always spelled the same.
into its internal structure. Several critiques of these black box approaches have appeared in the literature \cite{15, 17}, and some of these include the fact that they are applicable very late in the life cycle of the software, ignore information about testing & reliabilities of the components of which the software is made, and do not take into consideration the architecture of the software. With the widespread use of object oriented systems design and development, the use of component-based software development is on the rise. The software components can be commercially available off the shelf (COTS), developed in house, or developed contractually. Thus, the whole application is developed in a heterogeneous (multiple teams in different environments) fashion, and hence it may be inappropriate to model the overall failure process of such an application using one of several software reliability growth models (black box approach) \cite{3}. These heterogeneous systems, where components having different workloads and failure behaviors interact, have become more of a norm than an exception. Modern programs can no longer be treated as monolithic entities, but are likely to be made up of thousands or millions of little parts distributed globally, executing whenever called, acting as parts of one or more complex systems \cite{35}. Thus predicting the reliability of an application earlier in the life cycle, taking into account the information about its architecture, and the testing & reliabilities of its components, is essential.

Recent research efforts have been focused on the development of approaches to predict the reliability of a software application taking into account its architecture. Goseva-Popstojanova \textit{et al.} \cite{14} classify the existing architecture-based models into three broad categories: state-based, path-based, and additive. State-based models use the control graph to represent software architecture, and predict reliability analytically. Path-based models compute software reliability considering the possible execution paths of the program. The execution paths may be determined using simulation, by executing the application \cite{18}, or algorithmically \cite{16, 41}. Additive models assume that each component reliability can be modeled by a non-homogeneous Poisson process (NHPP), which leads the system failure process to be NHPP with cumulative number of failures & failure intensity functions that are the sums of the corresponding functions for each component \cite{40}. Additive models do not consider the architecture of the application explicitly. The broad classification of architecture-based software reliability models is shown in Fig. 1.

Among the three categories of architecture-based software reliability models, state-based models have been explored to a greater extent than the other two approaches. However, the development of state-based approaches has been mostly ad hoc with little or no effort devoted towards comparing & contrasting these models, and highlighting their commonalities & differences. Also, to the best of our knowledge, no attempt has been made to offer any insight regarding the inputs accepted by each model, and the outputs produced by each model. From the existing literature in this area, it is not clear to a practitioner as to which model may be used to predict application reliability in each phase of its life cycle.

In this paper, we establish a unifying framework for state-based models for architecture-based software reliability prediction. We outline the information necessary for the specification of the state-based models of the software to predict its reliability. We also propose a systematic classification scheme for state-based approaches to reliability prediction. The classification scheme considers three aspects while categorizing the models:

1) the model used to represent the architecture of the application,
2) the model used to represent the failure behavior of its components, and
3) the solution method.

We place the existing state-based models in the literature in their appropriate categories according to the above three aspects. We then present an exhaustive analysis of those state-based models where the architecture of the application is modeled either as a discrete time Markov chain (DTMC), or a continuous time Markov chain (CTMC). We illustrate the DTMC- and CTMC-based models with examples. We also provide a detailed discussion regarding the input parameters required by each model, and how these parameters may be estimated from the different software artifacts. Depending on the software artifacts available during a given phase of the software life cycle, and the parameters estimated from these artifacts, we provide guidance regarding which model may be appropriate to predict the reliability of an application during each phase of the software life cycle.

The layout of the paper is as follows. Section II discusses the specification & solution methods of the state-based models. Section III presents an exhaustive overview of those state-based models for reliability prediction, where the architecture of the application is represented either using a DTMC or a CTMC. Section IV illustrates the DTMC- and CTMC-based models presented in Section III with examples. Section V summarizes the input parameters expected by each model, shows how these parameters may be estimated using different software artifacts, and provides a guideline regarding the selection of a suitable model for each phase of the software life cycle. Section VI presents conclusions, and directions for future research.

II. SPECIFICATION & SOLUTION METHODS OF STATE-BASED MODELS

The specification of state-based models to predict the reliability of an application based on its architecture requires knowledge about several features.
• **Architecture of the application:** This is the manner in which different components of the software interact, and is given by the intercomponent transition probabilities. Interaction is assumed to occur only by transfer of execution control. The intercomponent transition probabilities will depend on the operational profile of the software. Recent experimental studies indicate that the intercomponent transition probabilities may also depend on the number of faults present in the components of the software [13]. The architecture may also include information about the execution time (mean, variance, distribution) of each component. The architecture of the application can be modeled either as a DTMC [1], a CTMC [20], a SMP (Semi-Markov Process) [19], a DAG (Directed Acyclic Graph) [39], or a SPN (Stochastic Petri Net) [32]. The state of the application at any time is given by the component executing at that time, and the state transitions represent the transfer of control among the components. DTMC, CTMC, and SMP can be further classified into irreducible, and absorbing categories, where the former represents an infinitely running application, and the latter a terminating application, or one which operates on demand. The irreducible models can be solved to compute the state probabilities, the probability that a given component is executing at a particular time, or in the steady state. Analysis of the absorbing DTMC can provide the average number of times a component is executed in a typical run of the application, or the utilization of a component. Similarly, the absorbing CTMC & SMP can be analyzed to give the expected number of visits to a component during a typical execution of the application, the average execution time in a component during a typical run, and the expected time to completion of the application. DTMC, SMP, and CTMC can be used to model a sequential application where, at each instant, control lies in one and only one of the components. SPN, and DAG can be used to model concurrent applications, and to predict their expected time to completion [30]. DAG is restricted to modeling concurrent applications with no loops. SPN can be used to model concurrent applications with loops.

• **Failure behavior of the components & interfaces:** Failure models need to address the failure behavior of the components & interfaces between the components. The failure behavior of a component can be specified either by the reliability of the component, a constant hazard rate , or a time-dependent failure intensity. We assume that component failures are independent of other components [28]. Thus, the probability of failure or the reliability of the component is the probability that it will fail during a single execution, whether it is used in isolation or in the context of any application. The time-dependent failure intensity could be the failure intensity associated with one of the software reliability growth models [3]. The three failure models can be viewed to form a hierarchy as far as the level of detail that can be incorporated, and the accuracy of the reliability predictions produced. The reliability obtained when the failure behavior of the components is modeled by their reliabilities has the lowest accuracy, because it essentially treats the component as a black box, and hence all the executions of the component during a typical run of the application are treated to be independent. Representing the failure behavior by a constant failure rate improves the accuracy of the reliability prediction over the previous case, because it can account for the total execution time of the component over multiple visits in a typical run of the application. In general, the more time the execution time in a component, the higher the probability of failure, and using a constant failure rate as the failure model can account for this fact. Representing the failure model by time-dependent failure intensity leads to the most accurate reliability predictions because it can account for testing & usage characteristics of the component either through the measurement of code coverage [10], or failure data. Code coverage measurements could provide an indication of the usage of the component during both testing, and operational phases, whereas failure data could provide an indication of the usage of the component during the testing phase of the component. Transitions among components could either be instantaneous, or there could be an overhead in terms of time. In either case, the interfaces could be perfect (that is, no failure), or subject to failure; and the failure behavior of the interfaces can also be described by the reliabilities, constant failure rates, or time-dependent failure intensities.

The architecture of the application can be combined with the failure behavior of the components & the interfaces into a composite model which can then be analyzed to predict the reliability of the application. We will refer to this method of reliability prediction as the “Composite Method”. The other possibility is to solve the architectural model, and superimpose the failure behavior of the components & the interfaces on to the solution of the architectural model to predict application reliability. We refer to this method as the “Hierarchical Method”. Both the composite method & the hierarchical method also enable the computation of various performance measures, such as the time to completion of the application. A detailed discussion of the advantages & disadvantages of the composite & hierarchical solution methods, and when it may be appropriate to use each one of these solution methods, is provided in Section V.

The information required for the specification of state-based models, and the different ways in which this information can be analyzed in order to predict reliability, is depicted in Fig. 2.

Table I provides a classification of the state-based approaches based on the combination of the architecture of the software, and the failure behavior of the components, which employ the “Composite Method” for their analysis, while Table II classifies the approaches which use the “Hierarchical Method” for their solution. In Tables I & II, the rows indicate the architectural model used to characterize the application, and the columns indicate the failure model used to describe the failure behavior of the components. Referring to Table I, the intersection of the first column & the first row indicates that the application architecture is described by an irreducible DTMC, and the failure behavior of the components of the application is given by the prob-

---

3The terms component, and module are used interchangeably in this paper.
abilities of failure, or reliabilities. Similarly A-DTMC, I-CTMC, A-CTMC, I-SMP, and A-SMP represent architectures modeled by absorbing DTMC, irreducible CTMC, absorbing CTMC, irreducible SMP, and absorbing SMP respectively.

The classification is based on the assumption that the interfaces among the components do not fail, and the transitions among the components are instantaneous. We note that these assumptions are not restrictive, and can be easily relaxed to incorporate interfaces which fail, or to include constant failure rates, or time-dependent failure intensities. However, although models can easily incorporate unreliable interfaces, very little information is available regarding the parameters used to estimate the failure behavior of the interfaces. The classification scheme thus characterizes every state-based model according to the architectural model, failure model of the components, and the solution method employed.

### III. RELIABILITY PREDICTION

In this section, we discuss various state-based models which can be used to predict the reliability of an application, when the architecture of the application is modeled by a DTMC or a CTMC. A detailed overview of DTMC and CTMC which discusses the metrics that can be obtained from their analysis can be found in [34].

In both DTMC- and CTMC-based models, each state represents the execution of a single component of the application. The techniques assume that the components fail independently, that is, the failure behavior of one component does not influence the failure behavior of any other component in the application. Some of these models have been presented in the literature, and are discussed here for the sake of completeness, as well as to compare & contrast them with other models. Each state-based model is characterized by three aspects, namely, the model used to represent the architecture of the application, the model used to represent the failure behavior of the components of the application, and the solution method used for reliability prediction.

#### A. DTMC-Based Models

**DTMC_1:**

Architectural Model: Absorbing DTMC (A-DTMC)

Failure Model: Reliabilities

Solution Method: Composite

This method has been discussed by Cheung [1], where he assumes that the reliabilities of the components are known, and the architecture of the application is modeled by an absorbing DTMC. The composite model, consisting of the architecture of
the application, and the failure behavior of its components, is represented by a transition probability matrix $P_C$, which is constructed as follows: A probability $p_{ij}$ is attached to every directed branch $(i, j)$, which is the probability that the transition $(i, j)$ is taken after the execution of component $i$. The reliability of node $i$ is assumed to be $R_i$. It is assumed that the application has a single entry node, and a single exit node, although this assumption can be relaxed to allow for multiple entry & exit nodes. Let the entry node be denoted by 1, which is the initial state of the DTMC. The terminal states $C$, and $F$ are added, which represent the state of correct output, and failure, respectively. For every node $i$, a directed branch $(i, F)$ is created with transition probability $1 - R_i$, representing the occurrence of a failure in the execution of component $i$. The original transition probability between $i$ & $j$ is modified to $R_i p_{ij}$, which represents the transfer of control to $j$ from $i$, conditional to the successful execution of $i$. A directed branch is created from the exit node $n$ to $C$, with transition probability $R_n$, which represents the correct termination at the exit node. Note that $p_{ij} = 0$ if the branch $(i, F)$ does not exist. The reliability of the application after $k$ steps is the probability of reaching the terminal state $C$, starting from the initial state 1, and is given by $p_{1C}(k)$.

**DTMC_2:**

Architectural Model: Absorbing DTMC (A-DTMC)

Failure Model: Reliabilities

Solution Method: Hierarchical

The reliability of the application when the architecture is represented by an absorbing DTMC, and the reliabilities of the components are known, using a hierarchical method of analysis, is given by

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

(1)

where $R_i$, and $V_i$ denote the reliability, and the expected number of visits to component $i$ respectively. $V_i$ can be obtained from the analysis of the original absorbing DTMC describing the application architecture. We note that $V_i$, which denotes the expected number of visits to component $i$, can be fractions. The expected number of visits to each component enables us to capture the effect of infinite execution paths that might exist due to the presence of loops. An execution path may be defined as the sequence of components that are executed; the first component in the sequence is the start component, and the last component in the sequence is the exit or the termination component of the application. All the components along a path form a series system from the point of view of reliability. The reliability of the application for a particular execution path is then computed as a product of the reliabilities of the components along that path. The application reliability is computed by averaging across all path reliabilities. However, this approach of enumerating the paths does not account for infinite paths. Because of the ability to capture an infinite number of execution paths, state-based approaches provide more accurate reliability predictions as compared to path-based approaches, which enumerate all the possible execution paths in an application [18].

An additional advantage of the state-based approach is that it can also be used for performance analysis, to compute the expected completion time of the application [34]. If $t_i$ denotes the expected execution time in a component $i$ per visit, then the expected execution time $\bar{t}$ of the application is given by

$$ \bar{t} = \sum_{i=1}^{n} V_i t_i $$

(2)

**DTMC_3:**

Architectural Model: Absorbing DTMC (A-DTMC)

Failure Model: Constant failure rates

Solution Method: Hierarchical

The reliability of the application, when the architecture is modeled by an absorbing DTMC, and a constant failure rate is associated with each of the components, is given by

$$ R = \prod_{i=1}^{n} (e^{-\lambda_i t_i})^{V_i} = \prod_{i=1}^{n} e^{-\lambda_i V_i t_i} $$

(3)

where $V_i$ denotes the expected number of visits to component $i$ in a typical run of the application, and $t_i$ is the expected execution time in component $i$ per visit.

**DTMC_4:**

Architectural Model: Absorbing DTMC (A-DTMC)

Failure Model: Time-dependent failure intensities

Solution Method: Hierarchical

When a time-dependent failure intensity is associated with each component, and the architecture of the application is modeled by an absorbing DTMC, its reliability is given by

$$ R = \prod_{i=1}^{n} e^{-\int_{0}^{V_i t_i} \lambda_i(\tau) d\tau} $$

(4)

where $V_i$ & $t_i$ are as defined in the case of DTMC_3. $V_i t_i$ is thus the expected total execution time in component $i$ per run of the application. Modeling the failure behavior of a component using time-dependent failure intensity enables us to accommodate the effect of intra-component dependence, that is, dependence that arises due to the multiple executions of a component, such as in the case of a loop [9]. Modeling the failure behavior of a component using a constant failure rate does not allow us to capture such intra-component dependence, and in general leads to pessimistic predictions of reliability. In [18] the authors model the failure behavior of individual components using their reliabilities, and suggest collapsing multiple executions of a component into $k$ components, where $k$ is defined as the degree of independence.

4The other type of dependence is inter-component dependence, which is the influence that the execution of one component exerts on the failure behavior of the other component.
**DTMC_5:**

Architectural Model: *Irreducible DTMC (I-DTMC)*

Failure Model: *Reliabilities*

Solution Method: *Composite*

The composite model of an irreducible DTMC representing an infinitely running application, along with the reliabilities of the components, is constructed by adding a terminal state $F$ which corresponds to the failure of the application. The DTMC under consideration has a single absorbing state, and this state is eventually reached with probability 1. We can compute the expected number of time steps until application failure in this case.

**DTMC_6:**

Architectural Model: *Irreducible DTMC (I-DTMC)*

Failure Model: *Reliabilities*

Solution Method: *Hierarchical*

The reliability of an application, when the architecture is defined by an irreducible DTMC, and the failure model used for the components is the probability of failure or reliability, is given by

$$R = \sum_{i=1}^{n} \pi_i R_i$$  \hspace{1cm} (5)

where $\pi_i$ is the probability that the application is executing component $i$ in the steady state.

The mean number of time steps until failure can be computed as follows. In each time step, the probability of application failure is $(1 - R)$. The number of time steps until failure then follows a modified geometric distribution. The expected number of steps until failure, denoted by $MNTF$, is then given by [34]

$$MNTF = \frac{R}{1 - R}$$  \hspace{1cm} (6)

**DTMC_7:**

Architectural Model: *Irreducible DTMC (I-DTMC)*

Failure Model: *Constant failure rates*

Solution Method: *Hierarchical*

The overall failure rate of the application $\lambda$, when the architecture of the application is modeled by an irreducible DTMC, and a constant failure rate is associated with the components, is given by

$$\lambda = \sum_{i=1}^{n} \pi_i \lambda_i$$  \hspace{1cm} (7)

The reliability of the application denoted by $R(t)$ is given by

$$R(t) = e^{-\lambda t}$$  \hspace{1cm} (8)

**B. CTMC-Based Models**

In this section we present the CTMC-based models.

**CTMC_1:**

Architectural Model: *Absorbing CTMC (A-CTMC)*

Failure Model: *Constant failure rates*

Solution Method: *Composite*

The architecture of the software is described by a CTMC, by specifying an exponential distribution for the execution time per visit for each component, in addition to the inter-component transition probabilities. The terminal states $C_i$ and $F$, which represent successful, and abnormal termination of the application respectively, are added; and the infinitesimal generator matrix $Q$, corresponding to this composite model is constructed in the following manner. For every node $i$, a directed arc $(i, j)$ is created with rate $\mu_{ij}$, where the distribution of the execution time of component $i$, is given by $\text{exp}(\mu_i)$. And $p_{ij}$ is the probability that the control is transferred to component $j$ after the successful execution of component $i$. Thus $q_{ij} = \mu_{ij} p_{ij}$. An additional directed arc is created from every node $i$ to node $F$, with a transition rate $\lambda_i$, corresponding to the failure of component $i$. The original node $n$ is also denoted as node $C$, corresponding to the successful execution of the application. The composite model can then be solved to compute the probability of being in state $C$, corresponding to the successful execution of the application.

**CTMC_2:**

Architectural Model: *Absorbing CTMC (A-CTMC)*

Failure Model: *Constant failure rates*

Solution Method: *Hierarchical*

The reliability of the application when its architecture is represented by an absorbing CTMC, and the failure behavior of the components represented by a constant failure rate, is given by

$$R(t) = \prod_{i=1}^{n} e^{-\lambda_i L_i(t)}$$  \hspace{1cm} (9)

where $\lambda_i$ is the failure rate of component $i$, and $L_i(t)$ is the expected execution time in component $i$ during a typical run. The fundamental difference between (3) & (9) is that the total execution time in a component can be computed by solving the CTMC, whereas, in case of a DTMC, the expected execution time in a component per visit is specified separately, in order to determine the total execution time in the component.

**CTMC_3:**

Architectural Model: *Absorbing CTMC (A-CTMC)*

Failure Model: *Reliabilities*

Solution Method: *Hierarchical*

When the architecture of the application is represented by an absorbing CTMC, and the reliabilities of the components are known, the reliability of the application using the hierarchical method of analysis is given by (1).
When the architecture of the application is represented by an absorbing CTMC, and the failure behavior of the components is represented by time-dependent failure intensities, then its reliability is given by

$$R(t) = \prod_{i=1}^{n} e^{-\int_0^{t} \lambda_i(\tau) d\tau}$$

(10)

where $\lambda_i(\tau)$ denotes the failure intensity of component $i$, and $L_i(t)$ denotes the total expected execution time in component $i$.

**CTMC_5:**

Architectural Model: *Irreducible CTMC (I-CTMC)*

Failure Model: *Constant failure rates*

Solution Method: *Hierarchical*

The infinitesimal generator matrix in this case is constructed by adding a state $F$ corresponding to the failure of the application. However, because this represents an infinitely running application, there is no state corresponding to the successful execution of the application. The addition of the state $F$ transforms the irreducible CTMC into an absorbing one, with a single absorbing state. Given infinite time, the application will reach this absorbing state with probability 1. We can compute the mean time to absorption or mean time to failure (MTTF) for the application in this case. As far as the architectural model and the failure behavior of the components is concerned, the approach described by Rubino [21], [22] falls under this category. However, in their approach, the interfaces are assumed to be unreliable with failure behavior represented by constant failure rates.

**CTMC_6:**

Architectural Model: *Irreducible CTMC (I-CTMC)*

Failure Model: *Constant failure rates*

Solution Method: *Hierarchical*

The analysis here is also similar to the corresponding DTMC case, and has been discussed by Laprie & Kanoun [20]. We assume that the failure rates are small compared to the rates governing the transitions among the components; equivalently stated, a large number of transitions will take place among the components, before the occurrence of a failure. The overall failure rate of the application is given by (7), and the reliability of the application is given by (8).

**CTMC_7:**

Architectural Model: *Irreducible CTMC (I-CTMC)*

Failure Model: *Reliabilities*

Solution Method: *Hierarchical*

The analysis in this case is similar to that of its DTMC counterpart (CTMC_6), and the reliability of the application is given by (5). This approach has been discussed by Laprie & Kanoun [20].

Thus, we see that the composite method of analysis is feasible, only if the type of the model which is obtained by combining the architectural model with the component failure models is the same as the type of the original architectural model. For instance, composite analysis is possible if the architecture is modeled by a DTMC, and the reliabilities of the components are given, because the combination of the architectural model, and the failure models of the components results in a DTMC model. Similarly, in the case of a CTMC-based architecture, composite analysis is possible only if the failure behavior of the components is characterized by constant failure rates. Hierarchical analysis is possible when the architecture is described by an absorbing DTMC, and the failure behavior of the components is characterized by reliability, constant failure rate, or time-dependent failure intensity. Hierarchical analysis is also possible when the architecture is represented by an absorbing CTMC for all the three failure models. In the irreducible case, hierarchical analysis can be used to predict reliability when the components are characterized either by their reliabilities, or constant failure rates. The other advantages of composite & hierarchical solution methods, and when it might be appropriate to use each one of these methods, are discussed in Section V.

**IV. ILLUSTRATIONS**

In this section, we illustrate the DTMC- and CTMC-based approaches discussed in Section III with examples. We use a variant of the application reported in [13] as a running example in this section. The application provides a language-oriented user interface, which allows the user to describe the configuration of an array of antennas. Its purpose is to prepare a data file in accordance with a predefined format, and characteristics from a user, given the array antenna configuration described using an appropriate array definition language. The application was developed for the European Space Agency in the C language, and consists of approximately 10,000 lines of code. It is divided into three components, namely, the Parser, the Computational, and the Formatting. The architecture of the application was estimated by an analysis of the application source code, and is shown in Fig. 3. Components 1, 2, and 3 correspond to the Parser, Computational, and Formatting components respectively. Component E represents the terminating state of the application. The inter-component transition probabilities are reported in Table III. The reliabilities of the individual components were estimated via fault insertion testing, and are given in Table IV. The expected execution times for each component per visit, denoted by $t_i$, and measured using the UNIX utility clock, are reported in Table IV. $\mu_i$, which is the rate of departure from
state \( i \) of the CTMC, can be computed using the expected execution time per visit in each component \( t_i \), and is given by

\[
\mu_i = \frac{1}{t_i} \tag{11}
\]

The basic application is modified & enhanced to describe different combinations of the architectural models with the failure models presented in the previous section. To illustrate the models when the failure behavior of the components is characterized by constant failure rates, we compute the constant failure rate for component \( i \), denoted by \( \lambda_i \), from its reliability \( R_i \), and the expected execution time in component \( i \) per visit denoted by \( t_i \), using the expression [34]

\[
\lambda_i = \frac{1}{t_i} \ln R_i \tag{12}
\]

The constant failure rates for components 1, 2, and 3, computed using (12), are listed in Table IV.

To illustrate the approaches when the failure behavior is characterized by time-dependent failure intensities, we assume that the time-dependent failure intensity of every component is specified by the failure intensity of the Goel-Okumoto software reliability growth model [6]

\[
\lambda_i(t) = a_i b_i e^{-b_i t} \tag{13}
\]

In (13), \( a_i \) denotes the expected number of faults which would be detected from component \( i \) in infinite time, and \( b_i \) denotes the failure occurrence rate per fault. To estimate the reliabilities of the components, two faults each were reinserted in components 1 & 2, and component 3 was fault-free. As a result, we assume the value of \( a_1 \) & \( a_2 \) to be 2 faults, whereas the value of \( a_3 \) to be zero. Based on the value of \( a_i \), and \( t_i \) which is the expected execution time in component \( i \) per visit, we compute \( b_i \) using the expression

\[
b_i = -\frac{1}{t_i} \ln \left( 1 + \frac{1}{a_i} \ln(R_i) \right) \tag{14}
\]

\( b_i \) for components 1, and 2 computed using (14), were 0.0045, and 0.0146 respectively. \( b_3 \) is assumed to be zero. The failure intensities for components 1 & 2 are listed in Table IV.

The composite model of the application as per \( DTMC_1 \) is shown in Fig. 4. The application reliability when its architecture is represented by an absorbing DTMC, and the failure behavior of the components is given by their reliabilities, obtained using the composite method (\( DTMC_1 \)), is 0.6874. The reliability computation was performed using MATLAB. Based on the hierarchical method of analysis (\( DTMC_2 \)), the reliability computed using MATLAB is 0.6740. Using the hierarchical method, and the expected execution time in each component from Table IV, the expected time to completion of the application is 62.29 msecs. The reliability of the application when the failure behavior is characterized by constant failure rates (given in Table IV), using the hierarchical method of analysis as per \( DTMC_3 \), is 0.6740. The reliability of the application when the failure behavior is characterized by time-dependent failure intensities as shown in Table IV, using the hierarchical method of analysis as per \( DTMC_4 \), is 0.6740. The reliabilities in the cases of \( DTMC_3 \) & \( DTMC_4 \) are computed using MATLAB.

To illustrate the reliability prediction approaches for continuously running applications which are modeled by irreducible DTMC, we add a transition from state \( E \) to state 1, with the

<table>
<thead>
<tr>
<th>Component #</th>
<th>( R_i )</th>
<th>( \lambda_i )</th>
<th>( \lambda_i(t) )</th>
<th>Expected execution time ( t_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.8428</td>
<td>0.0086/msec</td>
<td>( 2.00 + 0.0045 + e^{-0.0045t} )</td>
<td>20msec</td>
</tr>
<tr>
<td>2</td>
<td>0.8346</td>
<td>0.0278/msec</td>
<td>( 2.00 + 0.0146 + e^{-0.0146t} )</td>
<td>6.5msec</td>
</tr>
<tr>
<td>3</td>
<td>1.0000</td>
<td>0.00</td>
<td>0.00</td>
<td>76.00msec</td>
</tr>
</tbody>
</table>

Fig. 3. Architecture modeled by an absorbing DTMC.

Fig. 4. Composite model as per \( DTMC_1 \).
transition probability $p_{E_1} = 1.0$. The architecture of the continuously running application modeled by an irreducible DTMC is shown in Fig. 5. The composite model as shown in Fig. 6 (as per DTMC_5) has a single absorbing state corresponding to the failure of the application, and the steady state probability of reaching the failure state is 1.00. The expected number of steps that the DTMC would take to reach the failure state in this case is 8.9000. The expected number of time steps until failure is computed using SHARPE [30]. The reliability of the application when the architecture is modeled by an irreducible DTMC, and the failure behavior of the components is characterized by their reliabilities using the hierarchical method of analysis as per DTMC_6, is 0.8966. The expected number of time steps until failure computed using (6) is 8.670. If the failure behavior of the components is given by constant failure rates as in Table IV, then the overall failure rate of the application using the hierarchical method of analysis as per DTMC_7 is 0.0110. The reliability of the application $R(t)$ in this case is given by $e^{-0.0110 t}$, with a mean time to failure (MTTF) of $1/0.0110 = 90.90$ msecs.

Because the expected execution time in each component per visit is known, the architecture of the application can also be modeled by an absorbing CTMC, as shown in Fig. 7. When the failure behavior of the individual components is characterized by their reliabilities, the reliability of the application using the hierarchical method of analysis, as per CTMC_3, is 0.6740. If the failure behavior of the components is given by their constant failure rates, then the reliability of the application based on the composite model shown in Fig. 8, as per CTMC_1, is 0.7063. Using the hierarchical method for the same scenario, as per CTMC_2, the reliability is 0.6740. When the failure behavior is characterized by time-dependent failure intensities, as shown in Table IV, the reliability of the application, as per CTMC_4, is 0.6740.

The absorbing CTMC is then transformed to an irreducible CTMC by adding the transition $p_{E_1} = 1.0$ as shown in Fig. 9. The rate $\mu_E$ was set to 100/msec for the purposes of illustration. This rate was set to be much higher than the rates of departure from the other states of the CTMC, so that the effect of the execution time in component $E$ before transitioning back to component 1 can be minimized. The reliability of the application, when the failure behavior of the components is given by component reliabilities using the hierarchical method of analysis as per CTMC_7, is 0.9011. If the failure behavior of the components is described by constant failure rates, then the overall failure rate of the application using the hierarchical method, as per CTMC_6, is 0.0078. The mean time to failure (MTTF) is 128.21 msecs. The composite model, as shown in Fig. 10, has a single absorbing state which represents abnormal termination of the application, and this state is eventually reached with a probability of 1.0. However, we can compute the mean time to failure (MTTF) of the application as per CTMC_5, which is 128.63 msecs. The MTTF computations are performed using SHARPE.
A summary of the reliability predictions obtained using DTMC approaches is given in Table V. A summary of the reliability predictions obtained using CTMC approaches is given in Table VI. Table V, and Table VI indicate that, when both hierarchical and composite analysis methods are possible, reliability (MTTF, and MNTF) predictions obtained using hierarchical analyses provide an approximation to the reliability (MNTF, and MTTF) predictions obtained using composite analysis. For example, referring to Table V, DTMC_1 employs a composite method of analysis to predict the reliability of a terminating application modeled by an absorbing DTMC with the failure behavior of individual components described by their reliabilities, while DTMC_2 employs a hierarchical method for the same combination of the architectural model, and the failure models. The reliability prediction obtained from the composite method (DTMC_1) is 0.6874, and the reliability prediction obtained from the hierarchical method (DTMC_2) is 0.6740. Similar observations can be noted in the case of the composite method (DTMC_5) (composite method), and DTMC_6 (hierarchical method). CTMC_1 & CTMC_2, and CTMC_5 & CTMC_6 exhibit similar properties.

V. MODEL SELECTION AND APPLICATION

In this section, we provide a detailed discussion about the input parameters necessary in order to apply each model, and how these parameters might be obtained from various software artifacts. The selection of a model to predict the reliability of an application in a given phase of its life cycle will depend on the software artifacts that are available for estimating the parameters during that phase.

Table VII summarizes the input parameters that are required in order to use each model described in Section III. In Table VII, “Yes” indicates that the parameter indicated in the particular column is needed for the model, “X” indicates that the particular parameter is not necessary in order to apply the model, “Mean” indicates that the mean execution time per visit is necessary for model use, and “Exponential” indicate that the execution time per visit to each component must follow an exponential distribution with a known mean. Table VII indicates that, in order to apply DTMC-based models, the transition probabilities among the components are needed. If the failure behavior of the components is modeled using a constant failure rate or a time-dependent failure intensity, then it is also necessary to know the mean execution time per visit to each component. The MTTF/MNTF-based models, however, do not impose a restriction on the distribution of the execution time per visit to each component; in fact, this distribution could be unknown. This is unlike the CTMC-based models, where in addition to the transition probabilities among the components, it is also necessary that the execution time per visit to each component follows an exponential distribution with a known mean.

Architecture-based analysis techniques may be employed in various phases of the software life cycle, starting from the design phase, up to and including the operational phase. The input parameters of the architecture-based models could be obtained
from different types of software artifacts depending on the phase of the software life cycle during which an architecture-based analysis is to be employed. However, it may not be possible to obtain all the parameters from all types of software artifacts. Due to this reason, the model that is most suitable for a given phase of the software life cycle will depend on the software artifacts that are available in that particular phase, and the parameters that can be estimated from these artifacts.

If architecture-based analysis is to be employed during the design phase, the inter-component transition probabilities can be estimated, either from extensive consultations with experts who are intimately familiar with the system, or from historical data from a prior release of the application or another similar application. The information could also be obtained from the occurrence probabilities of various scenarios based on the operational profile of the system, as illustrated in [41], or could be obtained from simulation. It is unlikely that either the mean or the distribution of the execution time per visit to each component can be determined in this phase. As a result, it may be appropriate to characterize the failure behavior of the components using their reliabilities. The reliabilities of the individual components can also be estimated, either from expert knowledge and opinion, or from historical data. If a component is picked off the shelf, then its reliability may be certified [36]. If employed in the design phase, architecture-based analysis enables an assessment of application reliability, an analysis of the sensitivity of the application reliability to the reliabilities of the individual components, and an exploration of tradeoffs among architectural alternatives [8], [37].

If architecture-based analysis is to be employed after the implementation of the application is partially or completely available, it may be possible to estimate the mean execution time in each component per visit. It may also be possible to determine whether the execution time per visit follows an exponential distribution. The estimation of the mean execution time, as well as the determination of the distribution of the execution time, could be based on the profile data collected during the execution of the application. Such profile data may be already collected for other purposes, such as coverage analysis [2], performance analysis [4], and program comprehension [5]; and could be reused for reliability prediction. If the execution time per visit to each component follows an exponential distribution, then CTMC-based models may be used. On the other hand, if the execution time per visit to each component does not follow an exponential distribution, then DTMC-based models may be appropriate. In these phases, due to the availability of the execution time information, we may be able to characterize the failure behavior of the components using a constant failure rate or a time-dependent failure intensity so that we can improve the accuracy of the reliability prediction obtained from the model. In the testing phase, if the failure data collected during the unit testing of the component is available, then these failure data could be used to estimate the parameters of the time-dependent failure intensity of the component. Alternatively, the parameters of the time-dependent failure intensity could be estimated using a combination of software metrics & code coverage as given by the Enhanced Non Homogeneous Poisson Process (ENHPP) model [10], [12]. Architecture-based analysis may be employed in the testing phase to determine the allocation of resources to each component so that the desired reliability target can be achieved in a cost-effective manner [23]–[25]. In the operational phase, the failure behavior of the components may be characterized using a time-dependent failure intensity, with the parameters of the failure intensity being estimated based on the code characteristics & the code usage. The failure behavior in the operational phase may also be characterized by a constant failure rate, and this constant failure rate may be estimated at the end of the testing phase. Architecture-based analysis may be used in the operational phase to identify components for reliability enhancement in order to achieve a maximum improvement in application reliability for a given level of resources. The analysis may also be used to assess the impact of porting the application from one platform to another.

Once a suitable model is selected, the next step is to choose the solution method to solve the model. The choice of the solution method is relevant only for those combinations of architectural & failure models where both the composite, and the hierarchical methods can be employed. The advantages & disadvantages of the composite & the hierarchical methods as discussed below must be carefully considered before a solution method is selected.

For a given combination of the architectural model, and the failure model, the composite method of analysis produces accurate reliability estimates. However, the composite method is cumbersome & inconvenient for the following reasons:

- An important utility of architecture-based analysis is in the design phase of the software, where various competing alternatives need to be evaluated; and when informed decisions regarding how many & which components should be developed in house, and which modules can be outsourced need to be made. Thus, in this phase, it is essential to assess the impact of an individual component on the overall application reliability. In the case of a composite model, to analyse such an impact, the combined model has to be

<table>
<thead>
<tr>
<th>Model</th>
<th>Architectural parameters</th>
<th>Failure behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTMC.1</td>
<td>Yes</td>
<td>Reliability</td>
</tr>
<tr>
<td>DTMC.2</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>DTMC.3</td>
<td>Yes</td>
<td>Constant failure rate</td>
</tr>
<tr>
<td>DTMC.4</td>
<td>Yes</td>
<td>Time-dependent failure intensity</td>
</tr>
<tr>
<td>DTMC.5</td>
<td>Yes</td>
<td>Reliability</td>
</tr>
<tr>
<td>DTMC.6</td>
<td>Yes</td>
<td>Constant failure rate</td>
</tr>
<tr>
<td>DTMC.7</td>
<td>Yes</td>
<td>Time-dependent failure intensity</td>
</tr>
</tbody>
</table>

Inputs Necessary for Model Application
re-constructed & re-solved to obtain revised predictions. This act involves repeated construction & solution of the combined model. This act is further exacerbated by the fact that, in practice, a software of moderate size can have several hundreds to thousands of states, which makes repeated construction & solution of the composite model both costly in terms of time as well as in processing power.

- The second problem arises due to the stiffness of the composite model. This is because of the fact that the probability of failure of the modules is quite low, compared to the transition probabilities among the components, which makes the transitions to the failure state unlikely. Solution techniques which take into account the stiffness will need to be employed in this case [26].

- Not all combinations of architectural and failure models are tractable analytically using the composite approach. For example, if the architecture of the application is represented by a discrete-time Markov chain, and the failure behavior of its individual components is represented by a constant failure rate, then a composite model based on these two pieces of information cannot be solved analytically or numerically.

The hierarchical approach produces an analytical application reliability function, which relates the application reliability to the reliabilities of its individual components, and the architectural statistics of the application. A unique feature of this application reliability function is that it facilitates sensitivity analysis, and optimizations. It also allows us to determine a ranking of the modules from the point of view of their criticality to application reliability, so that additional resources can be devoted to the reliability improvement of those modules which have the highest impact on application reliability. However, the reliability estimate produced by the hierarchical approach is an approximation of the estimate produced by the composite approach. Incorporating second-order architectural statistics can improve the accuracy of the reliability estimates produced by the hierarchical approach [7], [11].

Thus, when architecture-based analysis is being employed for the purposes of sensitivity analysis, exploration of architectural alternatives, and identification of bottlenecks, the hierarchical solution method is more desirable than the composite solution method. However, when accuracy of the reliability prediction obtained from architecture-based analysis is of paramount importance, the composite solution method must be employed.

VI. CONCLUSIONS, AND FUTURE RESEARCH

In this paper, we have proposed a unifying framework for state-based models applied to architecture-based software reliability prediction. We have outlined the information necessary for the specification of the state-based models of the software to predict its reliability. We have also developed a systematic classification scheme for state-based approaches to reliability prediction. The classification scheme considers three aspects while categorizing the models. These aspects include the model used to represent the architecture of the application, the model used to define the failure behavior of the components of the application, and the analysis method. We placed the existing state-based models in the literature in their appropriate categories. We then presented an exhaustive analysis of those state-based models where the architecture of the application is modeled either as a discrete time Markov chain (DTMC), or a continuous time Markov chain (CTMC). The DTMC- and CTMC-based models were illustrated using examples. We also provide a detailed discussion regarding the input parameters required by each model, and how these parameters may be estimated from the different software artifacts. Depending on the software artifacts that are available during a given phase of the software life cycle, and the parameters that can be estimated from these artifacts, we provide guidance regarding which model may be appropriate to predict the reliability of an application during each phase of its life cycle.

Our future research is focused along two dimensions. To enable the application of these models to real life software applications, different parameters of the models need to be estimated from various artifacts. We are presently developing techniques for parameter estimation based on different software artifacts. As discussed earlier, hierarchical methods provide reliability (MTTF, and MNTF) predictions which are approximations to the reliability (MTTF, and MNTF) predictions produced by composite models. However, hierarchical methods enable a concise representation of application reliability in terms of the failure behavior of the components, and architectural behavior which is captured by the expected number of visits to each component. This concise representation facilitates sensitivity analysis to identify the impact of individual components on application reliability. To exploit the advantage offered by hierarchical methods, namely ease of sensitivity analysis, the accuracy of the reliability predictions obtained using the hierarchical methods needs to be improved. Developing methods to enhance the accuracy of the reliability predictions obtained using hierarchical methods is another area of future research.

REFERENCES


Swapna S. Gokhale received the B.E. (Hons.) in Electrical and Electronics Engineering, and Computer Science from the Birla Institute of Technology and Science, Pilani, India in June 1994; and M.S., and Ph.D. in Electrical and Computer Engineering from Duke University in 1996, and 1998, respectively. Currently, she is an Assistant Professor in the Dept. of Computer Science and Engineering at the University of Connecticut. Prior to joining UConn, she was a Research Scientist at Telcordia Technologies in Morristown, NJ. She received a Best Paper Award in ISSRE'98. Her research interests include software reliability and performance, software testing, software maintenance, program comprehension and understanding, and wireless and multimedia networking.

Kishor S. Trivedi holds the Hudson Chair in the Department of Electrical and Computer Engineering at Duke University, Durham, NC. He has been on the Duke faculty since 1975. He is the author of a well known text entitled “Probability and Statistics with Reliability, Queuing and Computer Science Applications,” with a thoroughly revised second edition being published by John Wiley. He has also published two other books entitled “Performance and Reliability Analysis of Computer Systems,” published by Kluwer Academic Publishers, and “Queuing Networks and Markov Chains,” published by John Wiley. His research interests are in reliability and performance assessment of computer and communication systems. He has published over 300 articles, and lectured extensively on these topics. He has supervised 39 Ph.D. dissertations. He has made seminal contributions in software rejuvenation, solution techniques for Markov chains, fault trees, stochastic Petri nets, and performability models. He has actively contributed to the quantification of security and survivability. He is a Fellow of the Institute of Electrical and Electronics Engineers. He is a Golden Core Member of IEEE Computer Society. He is a co-designer of HARPE, SHAREP, and SPNP software packages that have been well circulated.