View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Andrews University

Andrews University

Digital Commons @ Andrews University

Master's Theses

Graduate Research

2014

A Hierarchical Framework for Estimating Heterogeneous Architecture-based Software Reliability

Wayne Morris
Andrews University

Follow this and additional works at: https://digitalcommons.andrews.edu/theses

Recommended Citation

Morris, Wayne, "A Hierarchical Framework for Estimating Heterogeneous Architecture-based Software Reliability" (2014). *Master's Theses*. 19.
https://digitalcommons.andrews.edu/theses/19

This Thesis is brought to you for free and open access by the Graduate Research at Digital Commons @ Andrews University. It has been accepted for inclusion in Master's Theses by an authorized administrator of Digital Commons @ Andrews University. For more information, please contact repository@andrews.edu.

Andrews University

Seek Knowledge. Affirm Faith. Change the World.

Thank you for your interest in the

Andrews University Digital Library

of Dissertations and Theses.

Please honor the copyright of this document by not duplicating or distributing additional copies in any form without the author's express written permission. Thanks for your cooperation.

ABSTRACT

A HIERARCHICAL FRAMEWORK FOR ESTIMATING HETEROGENEOUS ARCHITECTURE-BASED SOFTWARE RELIABILITY

by

Wayne Morris

Chair: Roy Villafane

ABSTRACT OF GRADUATE STUDENT RESEARCH

Thesis

Andrews University

College of Arts and Sciences

Title: A HIERARCHICAL FRAMEWORK FOR ESTIMATING HETEROGENEOUS ARCHITECTURE-BASED SOFTWARE RELIABILITY

Name of researcher: Wayne Morris

Name and degree of faculty chair: Roy Villfane, Ph.D.

Date completed: July 2014

Problem

The composite model approach that follows a DTMC process with constant failure rate is not analytically tractable for improving its method of solution for estimating software reliability. In this case, a hierarchical approach is preferred to improve accuracy for the method of solution for estimating reliability. Very few studies have been conducted on heterogeneous architecture-based software reliability, and those that have been done use the composite model for reliability estimation. To my knowledge, no research has been done where a hierarchical approach is taken to estimate heterogeneous architecture-based software reliability. This paper explores the use and effectiveness of a hierarchical framework to estimate heterogeneous architecture-based software reliability.

Method

Concepts of reliability and reliability prediction models for heterogeneous software architecture were surveyed. The different architectural styles were identified as batch-sequential, parallel filter, fault tolerance, and call and return. A method for evaluating these four styles solely on the basis of transition probability was proposed. Four case studies were selected from similar researches which have been done to test the effectiveness of the proposed hierarchical framework. The study assumes that the method of extracting the information about the software architecture was accurate and that the actual reliability of the systems used were free of software errors.

Results

The percentage difference in results of the reliability estimated by the proposed hierarchical framework compared with the actual reliability was 5.12%, 11.09%, 0.82%, and 52.14% for Cases 1, 2, 3, and 4 respectively. The proposed hierarchical framework did not work for Case 4, which showed much higher values in component utilization and therefore higher interactions between components when compared with the other cases.

Conclusions

The proposed hierarchical framework generally showed close comparison with the actual reliability of the software systems used in the case studies. However, the results obtained by the proposed hierarchical framework compared to the actual reliability were in disagreement for Case 4. This is due to the higher component interactions in Case 4 when compared with other cases and showed that there are limitations to the extent to

which the proposed hierarchical framework can be applied. The reasoning for the limitations of the hierarchical approach has not been cited in any research on the subject matter. Even with the limitations, the hierarchical framework for estimating heterogeneous architecture-based software reliability can still be applied when high accuracy is not required and not too high interactions among components in the software system exist.

Andrews University

College of Arts and Sciences

A HIERARCHICAL FRAMEWORK FOR ESTIMATING HETEROGENEOUS ARCHITECTURE-BASED SOFTWARE RELIABILITY

A Thesis

Presented in Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Wayne Morris

2014

©Copyright by Wayne Morris 2014
All Rights Reserved

A HIERARCHICAL FRAMEWORK FOR ESTIMATING HETEROGENEOUS ARCHITECTURE-BASED SOFTWARE RELIABILITY

A thesis
presented in partial fulfillment
of the requirements for the degree
Master of Science

by

Wayne Morris

APPROVAL BY THE COMMITTEE:

____
Roy Villafane, Ph.D., Chair

____
Stephen Thorman, Ph.D.

____
Rodney Summerscales, Ph.D.

______________________________ ______________________________
William Wolfer, M.S. Date approved

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF EQUATIONS

ACKNOWLEDGMENTS

I would first like to thank God for making this journey a successful one. His words have been my encouragement and guidance for overcoming the challenges throughout the pursuit of my endeavors. To my wife and child, Terrian and Gabrielle, who have been very encouraging and supportive in every way possible, I could not have asked for anything more. To my parents and family for believing in me and their prayers which have kept me strong and focused throughout this journey. Special thanks to Felicia, for her continuous spiritual encouragement, faith, and her very special assistance to my family. You have played a great role in the achievement of this milestone.

I would like to thank Dr. Villafane for doing a superb job as a teacher and an adviser who is always willing to give assistance. I would like to thank Prof. Wolfer for his encouragement, invaluable advice, and guidance which he has generously offered throughout the years. I would like to thank Dr. Thorman for his contributions to my development both academically and professionally. I would also like to extend thanks to the Department Chair, the entire ECS department, and the Andrews University community for the support, friends, and the various activities that were undertaken to help bolster our development while we continue to make progress and strides in our field.

I would like to also extend special thanks to friends: Dexter, Roberto, Jonathan, Rayon, Daci, Theo, Dueal, Veto, Jose, Nettleton, Denry, Ricardo and Melanie for all their support, encouragement, faith, and inspiring hope when all seems lost. Thank you all for your great support and contributions in making this journey a story of success.

CHAPTER 1

INTRODUCTION

Software reliability estimation is a critical activity of the software development process which must be understood to prevent or minimize the risks of software failures. It has continued to be an area of focus for developing new ways to measure, analyze, and predict failures so that preemptive actions can be taken in the software development process as well as to have in place well-established and prompt corrective response regime in the event of software failure.

There are several models in existence for carrying out software reliability analyses. In the earlier years of software development, software reliability analysis was mainly done using black box models which only provided information of the overall software reliability (Goseva-Popstojanova & Kamavaram, 2003). While these models were effective and very useful, it lacks the ability to provide information or analysis of the internal structure of software components (Goseva-Popstojanova & Trivedi, 2001). Hamlet (1992) and Horgan and Mathur (1996) have highlighted that black box models are applicable very late in the life-cycle of the software and ignore information about testing and reliabilities of the components that make up the software. In addition, black box models do not take into consideration the architecture of the software.

As the software process continued to evolve to its present state, most, if not all, software development practitioners have started using object-oriented approaches or incorporating commercially off the shelf (COTS) software components as part of the whole software system to enhance or add functionalities instead of completely redesigning or remodeling the software architecture (Goseva-Popstojanova & Trivedi, 2001). Using black box models for an object-oriented or component-based approach to building software system would require retesting the entire system to determine system reliability, which means that reliability analysis can be applied only very late in the life cycle of the software development process. However, with architecture-based approach, only information about the specific component that is being added or replaced is needed, and retesting of the entire system would not be necessary. Instead, a reassessment, based on the application of the reliability model, would be needed to predict the overall system reliability. This approach considers the software architecture and it is done by analyzing each component of the software architecture within a model framework for estimating the overall system reliability. It also allows for prediction of system reliability to be made based on formal, stochastic software models. Using these models, developers can identify critical software components and quantify their influence on the overall system reliability to optimize future testing activities (Koziolek, Schlich, & Bilich, 2010).

Architecture-based reliability modeling has gained substantial momentum and has shown much promise as documented in several researches. The approach for applying architecture based reliability has been classified into three categories: state-based, path-based, and additive (Goseva-Popstojanova & Trivedi, 2001), of which state-based has been the focus of earlier researches while path-based has become one of more recent interest. State-based approaches assume that the transfer of control between components has a Markov property which can be modeled as discrete time Markov chain (DTMC), Semi-Markov process (SMP), or continuous time Markov chain (CTMC). Path-based models compute software reliability considering the possible execution paths of the program and can be done experimentally or algorithmically. Additive models assume that each component can be modeled by a non-homogeneous Poisson process (NHPP) (Goseva-Popstanova, Mathur, & Kishor, 2001).

State-based approaches to estimating software reliability have been further classified into composite and hierarchical models (Gokhale & Trivedi, 2002). Composite models combine the architecture of the software and the failure behavior of its component as a single model, whereas the hierarchical model solves the architectural model and superimposes the failure behavior of the modules and that of the interfaces onto the solution to predict reliability.

While several studies have focused on architecture-based software reliability, very few have considered software as a heterogeneous architecture. Wang, Wu, and Chen (1999) developed a framework for estimating reliability of heterogeneous software architecture based on the Cheung (1980) model. Cheung's model is one of the most used state-based and composite models for estimating architecture-based software reliability and takes the operational profile into account by utilizing transition probabilities from one component to another. Wang, Wu, and Chen (1999) highlighted that the Cheung model lacked the flexibility to be applied to different architectural styles and therefore could pose a challenge for practitioners to configure the architecture to fit a particular reliability framework that best meets quality demand. Gokhale (2002) also stated that the composite model that follows a DTMC process with constant failure rate was analytically intractable as it combines both reliability and failure behavior as a single component which makes the solution to the model difficult when high accuracy is required. As a result, the hierarchical approach is preferred when the desire is to improve upon the accuracy of the method of solution for reliability prediction.

As several different architectural styles continue to emerge (Shaw, 1993) and given the heterogeneous nature of software system components, it is important to consider the different architectural styles as performance and availability could be impacted based on the architectural styles of the system (Wang, Chen, & Tang, 1999). It is also important that a reliability framework not only takes the heterogeneity of the architecture into account but is also analytically tractable for improving accuracy of the method of solution for reliability prediction. This is a deficiency in the model developed by Wang, Wu, and Chen (1999) as it is a composite model based on the model developed by Cheung (1980). As a result, exploring a hierarchical framework for estimating reliability in systems of heterogeneous architectural styles would be a basis for presenting more flexible ways of improving the accuracy of reliability models than with the composite model approach. It is much easier for the hierarchical approach to be represented as SMP or CTMC processes which also takes failure behavior, failure intensity, and time dependency into consideration (Gokhale & Trivedi, 1997).

Given the lack of flexibility for analysis observed in the model developed by Wang, Wu, and Chen (1999), the aim of this research was to explore a hierarchical framework for estimating and analyzing heterogeneous architecture-based software reliability. The successful application of the hierarchical framework would present a platform not only for more in-depth analysis of software reliability but also exploits its flexibility for considering additional factors which the composite model by Wang, Wu, and Chen (1999) would not otherwise consider.

Purpose of Study

This study aims at applying a hierarchical framework to accurately estimate heterogeneous architecture-based software reliability as an enhancement to the model developed by Wang, Wu, and Chen (1999). The purpose of this study is as follows:

1. To find an alternative and simpler approach for estimating software reliability of heterogeneous architectural styles
2. To identify the limitations, if any, of applying the hierarchical framework to heterogeneous architectural styles
3. To validate the application of the hierarchical framework not only to heterogeneous architecture but of its general use for predicting reliability
4. Successful use of the hierarchical framework would set the platform where other models could be easily derived or parameters could be added for improving accuracy.

The hierarchical approach offers much more flexibility for mathematically expressing reliability and taking into consideration additional parameters for improving accuracy which would be difficult using the composite approach.

Contributions

The main contribution of this study is to provide an alternative for estimating reliability of heterogeneous architectural styles from which analytical expressions can be developed to improve accuracy and include additional parameters. Very few studies have tackled the subject of estimating software reliability for heterogeneous architectural styles, and those including Wang, Wu, and Chen (1999) have all used the composite approach as a method of solution. The hierarchical approach has been applied to other systems having a specific or strict architectural designed structure, but no research, to my knowledge, has been conducted where the hierarchical framework has been applied to heterogeneous architectural styles. A number of studies have mentioned the use of hierarchical approach for estimating reliability but none have explored the extent to which the model can be applied or the limitations to the type of system architecture for which it can be applied.

In addition, the hierarchical approach that most studies have applied usually either follow a SMP or CTMC process which can be tedious to interpret mathematically. The hierarchical approach taken in this research is one that follows a DTMC process and is much more easily applied than models based on the SMP or CTMC process. While the study utilizes the hierarchical framework based on the DTMC process, it still has the ability to include failure rate and time domain as parameters, which is the characteristic of SMP and CTMC process models.

This study aims to fill these gaps which have been observed and to identify areas of deficiency in the software reliability estimation for heterogeneous architectural styles from which improvements can be developed. Doing so will set the platform for further studies where the model can be either refined or enhanced for higher accuracy as necessary.

Thesis Organization

The thesis is comprised of five chapters. This first chapter contains background information on the motivation for conducting research in the area of architecture-based software reliability. The chapter provided an overview of the concepts and deficiencies in studies that have been conducted in this area as well as justifications for proposing an alternative approach to current models in existence. The chapter also highlighted the importance of the research and its contributions.

Chapter 2 outlines details of the related work which have been done on architecture-based reliability and provides details of the concepts and principles of architecture-based reliability models.

Chapter 3 outlines in detail the methodology for the methods and approach taken in applying the proposed hierarchical framework for estimating software reliability of heterogeneous architectural styles. It also provides a detailed description of the case studies which were used in this research.

Chapter 4 provides the detailed analysis of the results obtained, while chapter 5 provides a conclusion which summarizes the study. Following this chapter are the appendix and reference list.

CHAPTER 2

LITERATURE REVIEW

Related Work

Various studies have been done and several mathematical models have been proposed for assessing architecture-based software reliability. One of the earliest models in assessing architecture-based software reliability model was developed by Cheung (1980). This model considers the software reliability with respect to the module's utilization and its reliabilities. It assumes that the program flow graph of a terminating application has a single entry and a single exit node, and that the transfer of control can be described by an absorbing DTMC with a transition probability matrix of $P = [p_{ij}]$ (Goseva-Popstojanova & Trivedi, 2001). It is often used as a basis for accuracy and deriving other architecture-based reliability models (Gokhale & Trivedi, 2002). Despite its effectiveness and accuracy, the model's method of solution is not analytically tractable and is not very reliable when high accuracy is required (Gokhale, 2002).

As a result, the model developed by Gokhale and Trivedi (2002) takes into consideration the number of visits to each state based on transition probabilities while incorporating failure behavior and expected time spent in each state as parameters. This model assumes that the failure behavior follows a Poisson process and is derived by Taylor series expansion. Therefore, the value estimated for reliability is an approximate value and expected to be more accurate when higher orders of the Taylor series expansion are considered. Since the model by Gokhale (2002) follows a Poisson process, components are assumed to fail at constant failure rate.

Despite these enhancements made to the Cheung's architecture-based model, it still lacks the ability to provide a framework for representing different architectural styles. Wang, Wu, and Chen (1999) proposed a framework based on Cheung's model for applying different software architectural styles and highlighted the importance of refining the model developed by Cheung (1980) when considering heterogeneous and complex architectures. Wang, Wu, and Chen (1999) developed a reliability model based on the following architectural styles: batch-sequential/pipeline style, parallel/pipe filter style, fault tolerance, and call and return. These styles are said to be a general representation of the different software architectural styles. For example, a client-server style would represent that of a call and return, a hierarchical and layered style would be similar to batch-sequential/pipeline, a multiprocessor environment would be that of a parallel/pipe filter style, and database distribution network could be seen as that of a fault tolerance style. The results of using the heterogeneous architectural style proposed by Wang, Wu, and Chen (1999) were very comparable with that of the actual reliability of the software system that was used. It must be noted that the Cheung's approach is purely maintained when considering a batch-sequential/pipeline where both the failure behavior and reliability of components are accounted for together. However, with other architectural styles, variations will be observed and therefore the need to evaluate the different architectural styles for predicting overall reliability.

Even with the refinement of Cheung's approach, the model proposed by Wang, Wu, and Chen (1999) does not separate reliability from failure behavior and therefore takes a composite approach to estimating reliability. As a result, the model becomes very difficult to evaluate if parameters such as failure behavior and time spent in each state as parameters are to be considered. In this case, a hierarchical approach could make the derivation of other mathematical expressions more easily to improve the accuracy of the solution. Gokhale (2002) outlined the importance or relevance for the use of the hierarchical approach as follows:

1. To readily analyze sensitivity of system reliability to the reliabilities of its components
2. To analyze the sensitivity of system reliability to the structural statistics of the system
3. To rank the components of the system in the order of their importance from the system reliability perspective, and thus identify the components that are critical to the system
4. To determine the allocation of reliability to individual components so that the overall reliability target of the system is achieved
5. To readily identify and analyze performance bottlenecks within components to improve overall performance of the software.

This would make the process of evaluating system reliability to be extended beyond treating the system as a DTMC into treating the system more like a semi-Markov process or continuous time Markov Chain (CTMC) through which real-time software reliability can be evaluated. Further, with this approach, traditional reliability analysis such as mean time to failure (MTTF), mean time between failure (MTBF), among others could be conveniently incorporated in the software reliability analysis. The Wang, Wu, and Chen (1999) model for estimating reliability of heterogeneous software architecture does not offer that level of flexibility, which this study had sought to address.

Software Reliability

Software reliability is defined as the probability of failure-free software operation for a specified period of time in a specified environment. It is one of the attributes of software quality, a multidimensional property including other customer satisfaction such as functionality, usability, performance, serviceability, capability, installability, maintainability, and documentation (Michael, 1996). Software reliability is generally accepted as the key factor in determining software quality since failures within a system can be quantified.

According to Nikora and Lyu (1999), software reliability is defined mathematically as follows:

Let "T" be a random variable representing the failure time or lifetime of a physical system. For this system, the probability that it will fail by time "t" is:

$$F(t) = P[T \leq t] = \int_0^t f(x)dx$$

Eq. 1: Failure Rate Expression

The probability of the system surviving until time t is:

$$R(t) = P[T > t] = 1 - F(t) = \int_t^\infty f(x)dx$$

Eq. 2: Survival Rate Expression

Based on Eq. 2, failure rate, which is the probability that a failure will occur in the interval $[t_1, t_2]$ given that a failure has not occurred before time t_1, can be written as:

$$\frac{P[t_1 \le t \le t_2 \mid T > t_1]}{t_2 - t_1} = \frac{P[t_1 \le t \le t_2]}{(t_2 - t_1)P[T > t_1]}$$
$$= \frac{F(t_2) - F(t_1)}{(t_2 - t_1)R(t_1)}$$

And Hazard rate, the limit of the failure rate as the length of the interval approaches zero, can be written as:

$$z(t) = \lim_{\Delta t \to 0} \frac{F(t + \Delta t) - F(t)}{\Delta t R(t)}$$
$$= \frac{f(t)}{R(t)}$$

Eq. 3: Hazard Rate

This is the instantaneous failure rate at time t, given that the system survived until time t. The terms hazard rate and failure rate are often used interchangeably.

A reliability objective expressed in terms of one reliability measure can be easily converted into another measure as follows (assuming an "average" failure rate, λ , is measured):

$$\mu(t) = \lambda * t$$
$$MTTF = 1/\lambda$$
$$\lambda = 1/MTTF$$
$$R(t) = e^{-\mu(t)}$$

Eq. 4: Basic Reliability Expressions and Model

Where $\mu(t)$ is the failure intensity and MTTF is the mean time to failure.

Software Reliability Models

Software reliability models can be grouped into two categories: black box model and white box model or architecture-based approaches. The black box models tend to treat the software as a monolithic whole while the white box approach analyzes the system based on its individual components (Goseva-Popstojanova & Trivedi, 2001). The most used or more established black box software reliability models are classified into a group called software reliability growth model (SRGM). They use the observed failure information and predict future failures that reflect the growth of reliability. A broad classification of SRGMs is given in Table 1 (Chandran, Dimove, & Punnekkat, 2010). SRGMs are classified under three major groups: finite, infinite based on the total number of failures expressed in infinite time, and Bayesian models. Tools such as CASRE (Nikora, 2002) and SMERFS (Fair & Smith, 1988) are available for analyzing SRGMs. These models depend only on the number of failures observed or time between failures.

Table 1

Classification of Software Reliability Growth Models

Finite		**Infinite**	**Bayesian Model**
Exponential	*Weibull and Gamma*	*Exponential*	
Jelinski-Moranda	S-Shaped Reliability Growth	Geometric	Littlewood-Verrall
Shooman	Weibull	Musa-Okumoto Logarithmic	
Musa Basic execution time		Duane	
Goel-Okumoto Schneidewind			

Perugupalli (2004) stated that with the growing emphasis on reuse, an increasing number of organizations are developing and using software not just as all-inclusive applications, as in the past, but also as component parts of larger applications. This makes existing black box models clearly inappropriate to model such a large component-based system. Instead, there is a need for modeling technique which can be used to analyze software components and how they fit together. The goal of the white box approach is to estimate the system reliability, taking the information about the components of software into account (Goseva-Popstojanova & Trivedi, 2001). While the approach may be different in quantifying software reliability, the aim of both models is to capture the reliability of the system as a function of failure behavior.

Architecture-Based Software Reliability Models

There are a number of architecture based models in existence for estimating reliability. Goseva-Popstojanova and Trivedi (2001) have done a survey of the various models and provide details of the assumptions that are made when using particular models. The survey conducted by Goseva-Popstojanova and Trivedi (2001) also highlighted the major common requirements for estimating reliability using architecture-based approaches. These include: module identification, software architecture, failure behavior, and combining architecture with failure behavior.

In addition to the advantage of architecture-based approaches over black box approaches, Goseva-Popstanova et al. (2001) have rationalized the motivation to apply architecture based software reliability as follows:

1. Understanding how the system reliability depends on its component reliabilities and their interaction
2. Studying the sensitivity of the application reliability to reliabilities of components and interfaces
3. Guiding the process of identifying critical components and interfaces for a given architecture
4. Selecting an architecture that is most appropriate for the system under study.

Gokhale and Trivedi (2002) corroborated the ease to which information, based on the motivation to use architecture-based approach as stated by Goseva-Popstanova et al. (2001), could be analyzed based on how each component affects the overall reliability.

While architecture-based approaches for estimating reliability have been a major focus in recent times, extracting the architecture on how components interact with each other can be challenging based on the assumptions, parameters considered, and the uncertainty which exist in reliability estimation (Goseva-Popstojanova & Kamavaram, 2003). In addition, the assumptions made based on extracting the structure could also lead to inaccurate reliability estimation (Parnas, 1975). As a result, accurate depiction of the software structure is critical for accurate reliability estimation.

According to Goseva-Popstanova et al. (2000), architecture-based approaches have been proposed mostly by ad hoc methods without any relationship among them. As a result, relationships among models are not clearly established. However, most of the approaches which have been researched are state-based models, which follow a DTMC, SMP, or CTMC process. Attempts to improve these models have been made through developing new models or new framework upon which the model can be applied.

Markov Chain

All state-based models are assumed to follow a Markov process. The proposed framework in this study utilizes the DTMC process and therefore only details of the DTMC process are presented in this literature.

The definition of a Markov chain is as follows (Grinstead & Snell, 2006): Considering a set of states, S = {s_1, $s_{2,}$ $s_{3,}$ $s_{4\ldots}$ $s_{r\}}$ where the process starts in one of these states and moves from one state to the next. If the chain is currently in state s_i, then it moves to state s_j at the next step with a probability denoted by p_{ij} and this probability does not depend upon which states the chain was in before the current state. This statement can be expressed formally as (Markov Chain, 2014):

Let X_1, X_2, X_3, ...represent a sequence of random variables with Markov property.

Since the present state, the future and past states are independent, therefore

The Probability, $Pr(X_{n+1} = \varkappa \mid X_1 = \varkappa_1, X_2 = \varkappa_2,\ldots, X_n = \varkappa_n) = Pr(X_{n+1} = \varkappa \mid X_n = \varkappa_n)$

If both sides of the equation are well defined.

The probabilities p_{ij} are called transition probabilities.

DTMC Application to Architecture-Based Software Reliability

In the DTMC process, both time and space are discrete. DTMCs can be classified into the following two categories (Gokhale, 2002):

1. Irreducible: A DTMC is said to be irreducible if every state can be reached from every other state
2. Absorbing: A DTMC is said to be absorbing, if there is at least one state i, from which there is no outgoing transition. A DTMC upon reaching an absorbing state is destined to remain there forever.

A DTMC is characterized by its one-step transition probability matrix, $P = [p_{i;j}]$. P is a stochastic matrix since all the elements in a row of P sum to one, and each element lies in the range [0; 1] (Gokhale & Trivedi, 2002). Since the architecture of the application follows the Markov Chain properties based on the formal definition presented, it means that components to be executed in the next state will depend only on the components of current state and the components of the next state will not have any dependency to the past history of the current state (Wang, Wu, & Chen, 1999).

Cheung Model

This model assumes that the architecture has a single entry node and a single exit node. Each node can be representative of a state or component. Let N_i and N_j represent an individual node where $0< i < j$ and R_i the reliability of N_i, and the probability of moving from N_i to N_j is the transition probability $N_{i,,j}$. Therefore, the reliability of successfully reaching state $N_{i,,j}$ is estimated as $R_iN_{i,,j.}$ Based on the state diagram, a transition matrix M can be defined for the value of $M(i,j)$ as follows:

$$\begin{array}{c|cccccccc|}
 & \boldsymbol{N_1} & \boldsymbol{N_2} & \boldsymbol{N_3} & & \boldsymbol{N_i} & \cdot\cdot & \boldsymbol{N_{n-1}} & \boldsymbol{N_n} \\
\boldsymbol{N_1} & 0 & R_1N_{1,2} & R_1N_{1,3} & \cdot\cdot & R_1N_{1,i} & \cdot\cdot & R_1N_{1,n-1} & R_1N_{1,n} \\
\boldsymbol{N_2} & R_2N_{2,1} & 0 & R_2N_{2,3} & \cdot\cdot & R_2N_{2,i} & \cdot\cdot & R_2N_{2,n-1} & R_2N_{2,n} \\
\boldsymbol{N_3} & R_3N_{3,1} & R_3N_{3,2} & 0 & \cdot\cdot & R_4N_{3,i} & \cdot\cdot & R_3N_{3,n-1} & R_3N_{3,n} \\
\boldsymbol{N_i} & R_4N_{4,1} & R_4N_{4,2} & R_4N_{4,3} & \cdot\cdot & 0 & \cdot\cdot & R_4N_{4,n-1} & R_4N_{4,n} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\boldsymbol{N_{n-1}} & R_{n-1}N_{n-1,1} & R_{n-1}N_{n-1,2} & R_{n-1}N_{n-1,3} & \cdot\cdot & R_{n-1}N_{n-1,i} & \cdot\cdot & 0 & R_{n-1}N_{n-1,n} \\
\boldsymbol{N_n} & R_nN_{n,1} & R_nN_{n,2} & R_nN_{n,3} & \cdot\cdot & R_nN_{n,4} & \cdot\cdot & R_nN_{n-1,n-1} & 0
\end{array}$$

Figure 1: Reliability Transition Matrix.

Let $N = \{ N_1 , N_2 , \ldots.., N_n \}$ be the set of states or components in the state diagram where N_1 is the initial state and N_n is the final state. $M^k (i, j)$ represents the probability of reaching state S_j from S_i through k transitions. Therefore, the reliability R beginning from S_i to S_j with total k transitions is represented as

$$R = M^k (i, j) \times R_j$$

Eq. 5: Reliability Evaluation by Transition Matrix

From initial state N_1 to final state N_n , the number of transitions k may vary from 0 to infinity, where 0 means that the initial state is also the final state and infinity means that a cyclic loop may occur indefinitely among the states. Therefore, it is necessary to consider every possible outcome of state transitions. Let T be a matrix such that:

$$T = I + M + M^2 + M^3 + \ldots\ldots = \sum_{k=0}^{\infty} M^k$$

Eq. 6: Expression for Finding Fundamental Matrix

Therefore, as k approaches infinity, it can be shown that it results in the fundamental matrix which can be expressed as:

$$T = (I - M)^{-1} = \sum_{k=0}^{\infty} M^k$$

Eq. 7: Fundamental Matrix Expression

Where I is the identity matrix.

From this expression, the overall system reliability is calculated as:

$$R = T(1,n) \times R_n,$$

Eq. 8: Cheung Method for Overall Reliability

where R_n is the reliability of the component of state n.

Wang, Wu, & Chen Model

This model utilizes the premises on which Cheung's model was developed by assuming that the architecture of the application follows a Markov's process. It combines both the reliability and failure behavior to determine overall system reliability. However, it goes further to evaluate the overall system reliability based on varying architectural styles. It takes into consideration four characteristics of the software architecture, namely, batch-sequential/pipeline style, parallel-pipe filter style, fault tolerance, and call and return style. These architectural styles are used to represent how software generally functions.

Mathematical models for calculating reliability of each architectural style were developed and are presented in the following sections.

Batch-Sequential/Pipeline Architectural Style

Both batch-sequential and pipeline styles are running in a sequential order. They share the same architecture view and state view. Figure 2 shows a diagram of the batch-sequential/pipeline style.

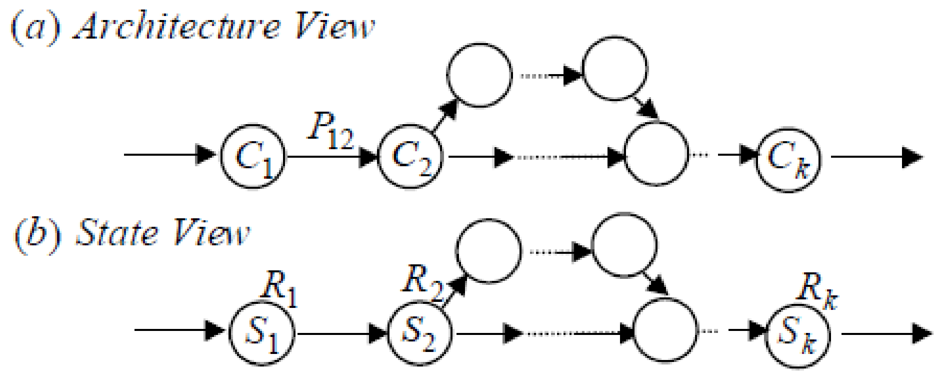

Figure 2: Batch-sequential and pipeline styles.

Assuming that the architecture is composed of *k* components, there will be *k* states in the Markov chain. The transition matrix *M* can be obtained as follows:

$$\begin{cases} M(i,j) = R_i P_{i,j} \ where\ S_i\ can\ reach\ S_j \\ M(i,j) = 0\ where\ \ S_i\ cannot\ reach\ S_j \end{cases} \quad \text{for } 1 \le i, j \le k$$

Where *M(i,j)* is the probability of successfully reaching state S_j from S_i.

The batch-sequential/pipeline architecture purely follows the process of estimating reliability using the Cheung's model as shown. An example of this architectural style is demonstrated in software that functions in performing one task at a time in a sequential manner.

Parallel/Pipe-filter style

In this architecture style, concurrent executions take place where components are running simultaneously as shown in Figure 3.

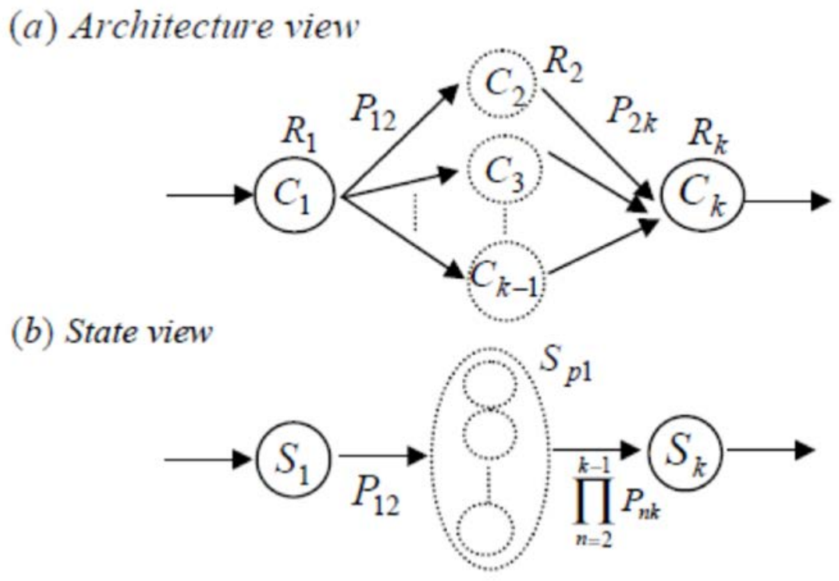

Figure 3: Parallel/pipe filter style.

The component reliabilities and transition probabilities are all independent of each other. As a result, the value of $M(\{S_{p1}\},\{S_k\})$ or the reliability of this architectural style is expressed as:

$$M(\{S_{p1}\},\{S_k\}) = \prod_{n=2}^{k-1} R_n P_{nk}$$

which is the product of all the component reliabilities in this state and the transition probabilities from components C_2 , C_3 , …, and $C_{k\text{-}1}$ to component C_k , respectively.

For *k* components, the transition matrix, taking into consideration the parallel/pipeline filter style that can be obtained is:

$$\begin{cases} M(i,j) = R_i P_{i,j} \; where \; S_i \; can \; reach \; S_j \\ M(i,j) = \prod_{c_n \, in \, S_i} R_n P_{nj,} \, S_i \in S_p \\ M(i,j) = 0 \; where \; N_i \; cannot \; reach \; N_j \end{cases} \quad \text{for } 1 \le i, j \le |S| \text{ and } 1 \le n \le k$$

An example of this architectural style can be found in a multiprocessor or multithreaded environment where more than one tasks are being done at the same time.

Fault Tolerance

A fault-tolerance architectural style consists of a primary component and a set of backup components, which may be implemented in different algorithms or data structures, from the primary component as shown in Figure 4.

These components, including the primary and the backups, are placed in parallel so that when one component fails, the others can still provide services.

It is assumed that all the backup components have the same transition probabilities as the primary component to each subsequent component. Let there be *k* components in which l=k-4 components are running as fault tolerance in the same state; therefore, the total number of states is k-l+1 according to Figure 4. By induction, entry $M(1,\{S_{b1}\})$ can be expressed as:

$$M(1,\{Sb1\}) = R_2 + \sum_{n=3}^{k-3}\left(\prod_{m=2}^{n-1}(1-R_m)\right)\mathrm{R_n}$$

To represent the transition matrix with fault tolerance architectural style, considering *k* components can be expressed as:

$$\begin{cases} M(i,j) = R_i P_{i,j} \text{ where } S_i \notin S_j \\ M(i,j) = R_{a1} + \sum_{q=a2}^{\mathrm{ar}} \left(\prod_{m=a2}^{q-1}(1-R_m)\right) R_n , \\ \quad S_i \in S_b \text{ and } S_i \text{ includes } C_{a1} \text{ to } C_{ar} \\ M(i,j) = 0 \text{ where } S_i \text{ cannot reach } S_j \end{cases}$$

for $1 \le i, j \le |S|$ *and* $1 \le a_r \le k$

An example of this architectural style can be found in a distributive system such as that of a distributive database system where there are back-ups to make the database continue to function even though one or more database components have failed. This type of system is especially important in transaction processing such as that used by the banks, hospitals, and airlines, among others.

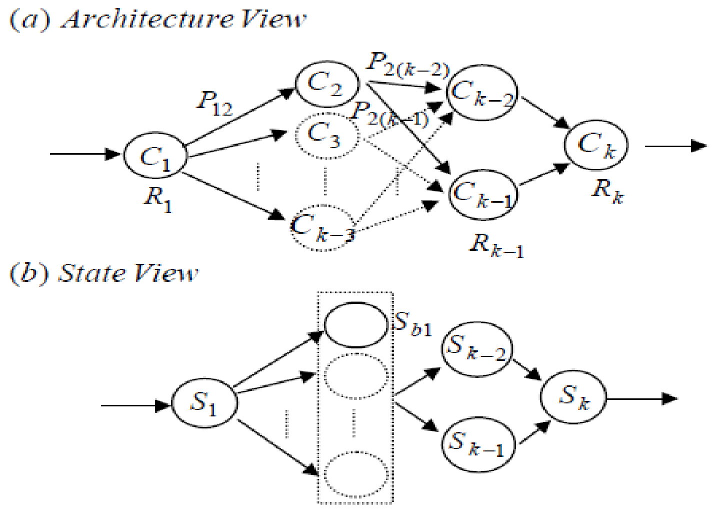

Figure 4: Fault tolerance.

Call-and-Return Architecture Style

In the call-and-return style, the execution of one component may request some services provided by the other components before transferring its complete control authority to others. Thus, after such a request is fulfilled by the called components, the control still returns to the calling component and executes the next statement from where the component left. Therefore, the called components may execute multiple times with only one time execution of the calling component. This is shown in Figure 5.

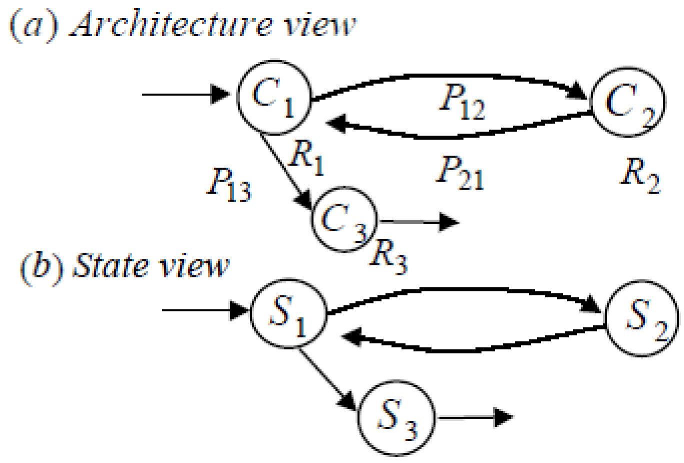

Figure 5: Call-and-return architectural style.

The expression for the transition matrix representing the call-and-return architectural style assuming that the total number of states is *k* with *k* components is:

$$\begin{cases} M(i,j) = R_i P_{i,j}\,, & S_i \; can \; reach \; S_j \\ M(i,j) = P_{ij}, & S_i \; can \; reach \; S_j \; and \; S_j \; is \; a \; component \\ \quad for\; 1 \le i,j \le k & \\ M(i,j) = 0, & S_i \; cannot \; reach \; S_j \end{cases}$$

An example of this architectural style can be found in web services where there is a server/client interaction. The client makes a request to server and the server returns a response to the client's request.

Estimating Overall Reliability

Considering the four architectural styles that have been presented by Wang, Wu, and Chen (1999), the framework for estimating system reliability of heterogeneous architecture is as follows:

$$\begin{cases} M(i,j) = 0 \; where \;\; S_i \; cannot \; reach \; S_j \\ M(i,j) = R_i P_{i,j} \; where \; S_i \; \notin S_p \; and \; S_i \; \notin S_b \\ M(i,j) = \prod_{c_n \; in \; S_i} R_n P_{nj,} \; S_i \; \in \; S_p \\ M(i,j) = R_{a1} + \sum_{q=a2}^{\mathrm{ar}} \left(\prod_{m=a2}^{q-1} (1 - R_m) \right) \mathrm{R_n} \, , \\ \qquad\qquad S_i \; \in S_b \; and \; S_i \; includes \; C_{a1} \; to \; C_{ar} \\ M(i,j) = P_{i,j} \, , S_j \; is \; called \; a \; component \end{cases}$$

for $1 \leq$ i, j, ar, n $\leq$ k

Gokhale & Trivedi Model

This model of estimating architecture-based software reliability utilizes transition probabilities between states as that of the Cheung model but goes further by including the expected number of visits per component, failure behavior (constant failure rate) and time as parameters. The assumptions are the same as Cheung's which states that the structure of the system follows an absorbing DTMC and the system consists of *n* components, and has a single initial state denoted by 1, and a single absorbing or exit state denoted by *n* (Gokhale, 2002; Gokhale & Trivedi, 2002). However, unlike the Cheung model, Gokhale and Trivedi (2002) is a hierarchical model that superimposes failure behavior on reliability.

The parameters considered for determining reliability in the Gokhale and Trivedi (2002) model are transition probability, $p_{i,j}$, number of visits per component, $X_{i,,j}$, time spent per visit per component, τ, and constant failure rate, λ, which are assumed to be known along with the reliability of each component and the structure of the system.

If there are *n* components that make up the software system and the reliability and the expected number of visits to each component are known, then the Gokhale and Trivedi (2002) model for calculating the overall reliability of the system is as follows:

$$\mathrm{R} = \prod_{i=1}^{n} R_i^{X_{i,j}}$$

Eq. 9: Gokhale Expression for Overall Reliability

Where R_i is the reliability of the component and *R* is the overall system reliability.

Eq. 9 can be expanded and be made more accurate by the Taylor series expansion. Thus Eq. 9 is a first order Taylor series expression. This is one of the drawbacks of the hierarchical approach where it becomes more accurate when higher order expressions of the Taylor series are used. With higher order Taylor series expansion of Eq. 9, mathematically it becomes more tedious to compute, thus making reliability determination more impractical. However, Goseva-Popstanova et al. (2001) showed that using the first order Taylor series expansion of the Gokhale hierarchical approach produced accurate reliability results when compared to the actual reliability. This was also corroborated by close accurate results which were obtained by Gokhale and Trivedi (2002) and Goseva-Popstojanova, Hamill, and Perugupalli (2005).

Gokhale and Trivedi (2002) showed how estimating the reliability using hierarchical approach, as expressed, gave comparable results when compared to that of Cheung (1980), a composite approach. The overall reliability of the system architecture in Figure 6 was calculated using the Cheung (1980) and Gokhale and Trivedi (2002) method. The component reliabilities and transition probabilities for are shown in Table 2 and Table 3 respectively.

The overall reliability was estimated to be 0.8299 for the Cheung's approach and 0.8264 for the Gokhale and Trivedi (2002) approach using the first order Taylor series expression. However, the Gokhale approach became more accurate and produced 0.8280 for the overall reliability when second order Taylor series expression, which included the variance of the expected number of visits, is used. The second order Taylor series expression for Eq. 9 can be expressed as:

$$R \;=\; \prod_{i=1}^{n}(R^{X_{i,j}} + \frac{1}{2}(R^{X_{i,j}})(logR_i)^2\,\sigma_{1,j}^2)$$

Eq. 10: Gokhale Expression for Overall Reliability using Second Order Taylor Series Expansion

Where $\sigma_{1,j}^2$ is the variance of the expected number of visits and $logR_i$.is the natural log of the reliability of component *i*.

The variance can be obtained from the fundamental matrix, as expressed in Eq. 6, as follows:

$$\sigma_{1,j}^2 = M(2M_{dg} - I) - M_{sq}$$

Eq. 11: Expression for Variance of the Expected Number of Visits, $\boldsymbol{\sigma_{1,j}^2}$

Table 2

Reliability of Component in Figure 6

Component	**Reliability**
1	0.999
2	0.980
3	0.990
4	0.970
5	0.950
6	0.995
7	0.985
8	0.950
9	0.975
10	0.985

Table 3

Transition Probabilities of Components in Figure 6

$p_{1,2}$ = 0:60	$p_{1,3}$ = 0.20	$p_{1,4}$ = 0.20	
$p_{2,3}$ = 0.70	$p_{2,5}$ = 0.30		
$p_{3,5}$ = 1.00			
$p_{4,5}$ = 0.40	$p_{4,6}$ = 0.60		
$p_{5,7}$ = 0.40	$p_{5,8}$ = 0.60		
$p_{6,3}$ = 0.30	$p_{6,7}$ = 0.30	$p_{6,8}$ = 0.10	$p_{6,9}$ = 0.30
$p_{7,2}$ = 0.50	$p_{7,9}$ = 0.50		
$p_{8,4}$ = 0.25	$p_{8,10}$ = 0.75		
$p_{9,8}$ = 0.10	$p_{9,10}$ = 0.90		

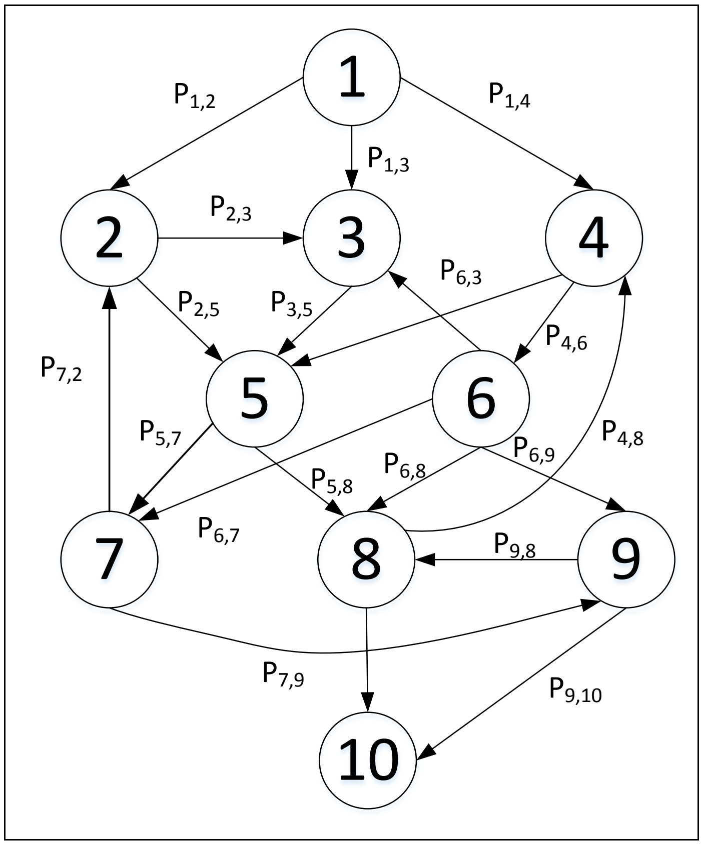

Figure 6: System Architecture represented by its components that follow an absorbing DTMC process.

A comparison of the reliability computed for each component with and without the consideration of the variance of expected number of visits is shown in Table 4. When variance is considered, the accuracy of reliability is slightly improved when compared to the composite method and therefore first order Taylor expression for the Gokhale and Trivedi (2002) model is satisfactory for providing estimation when high accuracy is not desired.

Table 4

Reliability Calculated Based on Gokhale and Trivedi Model Considering the Variance of the Expected Number of Visits

Component	**Reliability**	$\boldsymbol{R^{X_{i,j}}}$	$\boldsymbol{R^{X_{i,j}}}$ **with** $\boldsymbol{\sigma^2_{1,j}}$	$\boldsymbol{\sigma^2_{1,j}}$
1	0.999	0.99900	0.99900	0.00000
2	0.980	0.98183	0.98196	0.64437
3	0.990	0.99089	0.99092	0.54991
4	0.970	0.98734	0.98752	0.39284
5	0.950	0.93308	0.93396	0.71853
6	0.995	0.99874	0.99875	0.23185
7	0.985	0.99074	0.99081	0.62613
8	0.950	0.95618	0.95671	0.42252
9	0.975	0.99035	0.99043	0.24620
10	0.985	0.98500	0.98500	0.00000

Reliability Prediction Based on Time

One of the advantages of using the hierarchical approach in estimating reliability is that the model can be transformed into a SMP with the additional parameter of time. Gokhale and Trivedi (2002) developed an expression for evaluating performance analysis based on the Eq. 9.

Considering the reliability expression (Nikora & Lyu, 1999):

$R(t) = e^{-\mu(t)}$ where *μ(t)* represents failure rate as a function of time.

Since the assumption of constant failure rate is made, we can express *μ(t) as* λ. Therefore, the reliability can be expressed in terms of failure rate, λ, time spent per component, τ, and expected number of visit per component, $X_{i,j}$, as*:*

$$\mathrm{R} = \prod_{i=1}^{n} R_i^{X_{i,j}} = \prod_{i=1}^{n} e^{-\lambda\tau X_{i,j}}$$

Eq. 12: Overall Reliability Expression with the inclusion of Time and Failure Rate

Based on Eq. 12, Eq. 10 can be rewritten with consideration of second order Taylor expression as follows:

$$R \;=\; \prod_{i=1}^{n}\left(e^{-\lambda\tau X_{i,j}} + \frac{1}{2}\lambda^{2}\, e^{-\lambda\tau X_{i,j}}\, \tau_i^2\, \sigma_{1,j}^2\right)$$

Eq. 13: Second Order Taylor series Expression for Reliability with Failure Rate and Time

While the study is not focused on taking an SMP approach to estimating reliability, it is important to show the flexibility of the hierarchical approach and the extent to which other mathematical expressions could be derived to improve reliability prediction. This was what Gokhale (2002) meant by stating that the hierarchical approach is more analytically tractable as it is much easier to extract other information or analysis such as performance and sensitivity than the composite method to perform more accurate and in-depth reliability analysis.

Extracting the Software Architecture

Accurately extracting the software architecture is important for estimating reliability using architecture-based approaches (Parnas, 1975). The architecture can be extracted from the design phase by expert consultations or from prior release of the application. This approach is intended only if the architecture-based reliability analysis is to be carried out from the design stage of the software development life cycle (Gokhale & Trivedi, 2002). Yacoub, Cukic, and Ammar (1999) have shown that information on the architecture could also be obtained from the occurrence probabilities of various scenarios based on the operational profile of the system in the design phase.

Often, information on the architecture is not available and therefore has to be extracted from the source code of the application (Gokhale, 2002). This could be done using profilers (Fenlason & Stallman, 2013) or test coverage tools (Team, 1998). Component reliability can be obtained using failure data collected during unit testing of each component to estimate the parameters of the software reliability growth model (Farr, 1996). Non-failed executions and failures in the validation phase could also be used to estimate component reliabilities when information on the software architecture is scarce (Miller et al., 1992).

CHAPTER 3

HIERARCHICAL FRAMEWORK FOR ESTIMATING HETEROGENEOUS ARCHITECTURE-BASED SOFTWARE RELIABILITY

Introduction

The approach taken in applying the proposed hierarchical framework to estimate software reliability of heterogeneous architectural styles involves applying the concepts of the model proposed by Wang, Wu, and Chen (1999) and Gokhale and Trivedi (2002). Wang, Wu, and Chen (1999) described a composite approach and the concepts of analyzing the different architectural styles for reliability, while Gokhale and Trivedi (2002) described a hierarchical approach that applied to the number of expected visits to each component (component utilization) which could be used to transform the model into a SMP approach which included time and failure behavior as additional parameters for evaluating reliability. While this study is not focused on applying a SMP approach, the hierarchical approach remains important due to its flexibility and ability for other reliability expressions to be developed, which could change the entire perspective of software reliability analysis.

Four case studies were used in this study to explore the effectiveness of the proposed hierarchical framework. The data of these studies have been taken from previous studies and will be applied to the hierarchical framework being explored in this study. Comparative assessments of the results are provided between those obtained from the proposed hierarchical framework and actual results which have been obtained from the case studies. The assumptions and limitations of the study are also provided.

Information on the Software Architecture

Information on the architecture is needed to determine how the components of the software interact with each other. Component interactions are captured by the transition probabilities among its components. In many cases, this information is not known and can be extracted from various sources depending on the phase of the software cycle (Gokhale & Trivedi, 2002).

This study is not focused on extracting information about the architecture but assumes that the information is known. This information has already been given by the data which have been taken from the case studies. As a result, it is assumed that the methodology for extracting the information about the architecture is accurate and that all software errors have been found through the testing mechanism applied.

Reliability Metrics

Estimating overall reliability by architecture-based approaches requires that the reliability of each component is known. The reliability of components can be obtained by using failure data collected during the unit testing of each component to estimate the parameters of a software reliability growth model (Farr, 1996), or can be estimated by considering non-failed executions and failures in the validation phase (Miller et al., 1992). These activities are carried out in the operational phase of the software cycle where the extraction of information on reliability is done from the source code.

This study assumes that the reliability of each component is known and the reliability of each component has been given from the data obtained from the case studies under consideration. It is also assumed that the actual reliability, which our results will be compared with, is free of failure errors and that the methodology used for determining reliability is accurate and true.

My Approach

Our approach takes the four architectural styles identified by Wang, Wu, and Chen (1999) into consideration, namely, batch-sequential, parallel filter, fault tolerance and the call and return architectural style. The approach taken by Wang, Wu, and Chen (1999) involved combining the reliability with failure behavior for each architectural style through a DTMC process to establish a reliability transition matrix.

The main difference with my approach is that each architectural style is evaluated solely on the basis of the transition probability matrix from which the expected number of visits to each component or component utilization will be determined and applied to the heterogeneous software architecture. Therefore, a framework for evaluating the transition probabilities of each architectural style is developed to determine the component utilization so that the overall reliability can be found. Overall reliability is estimated by applying Eq. 7 to obtain the transition probability fundamental matrix and the Gokhale and Trivedi model as shown in Eq. 9.

The following sections of this study provide details of how the transition probability matrix is developed for finding the component utilization and subsequently estimating the overall reliability of the software system.

Evaluating the Transition Probability Matrix of Batch/Sequential Architectural Style

Given *n* components within a system that follow a batch/sequential architectural style, the architecture can be represented with its transitional probabilities as shown in Figure 7.

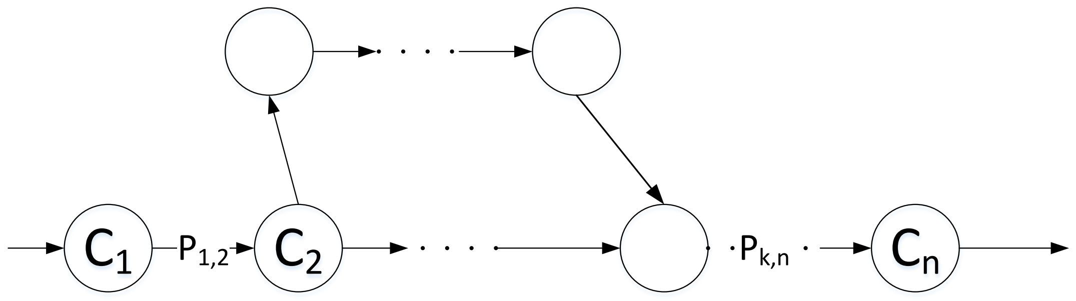

Figure 7: Batch/sequential architecture style.

Since a DTMC process is assumed, it therefore signifies that the stochastic process is purely maintained in this architecture which means that the transition probabilities within a row of the transition probability matrix sums to unity. As a result, the components within this architecture do not require any modifications since the execution is done in a sequential manner.

Evaluating the Transition Probability Matrix of Parallel/Pipe Filter Architectural Style

In the parallel filter architecture, components are executing concurrently. Since C_2, C_3 and C_{n-1} are executed concurrently by C_1 and C_2, C_3 and C_{n-1} concurrently execute C_n, C_2, C_3 and C_{n-1} can be represented by the same transition probability and therefore can be grouped collectively as a single component as shown by the demarcation in Figure 8.

As a result, the transition probability matrix can be transformed from the matrix (Figure 9) representing Figure 8 to the transition probability matrix in Figure 10.

The transition probability matrix can be evaluated for the component utilization (Eq. 7) and reliability (Eq. 9) with the assumption that the components functioning in the parallel structure are utilized equally at the same utilization rate.

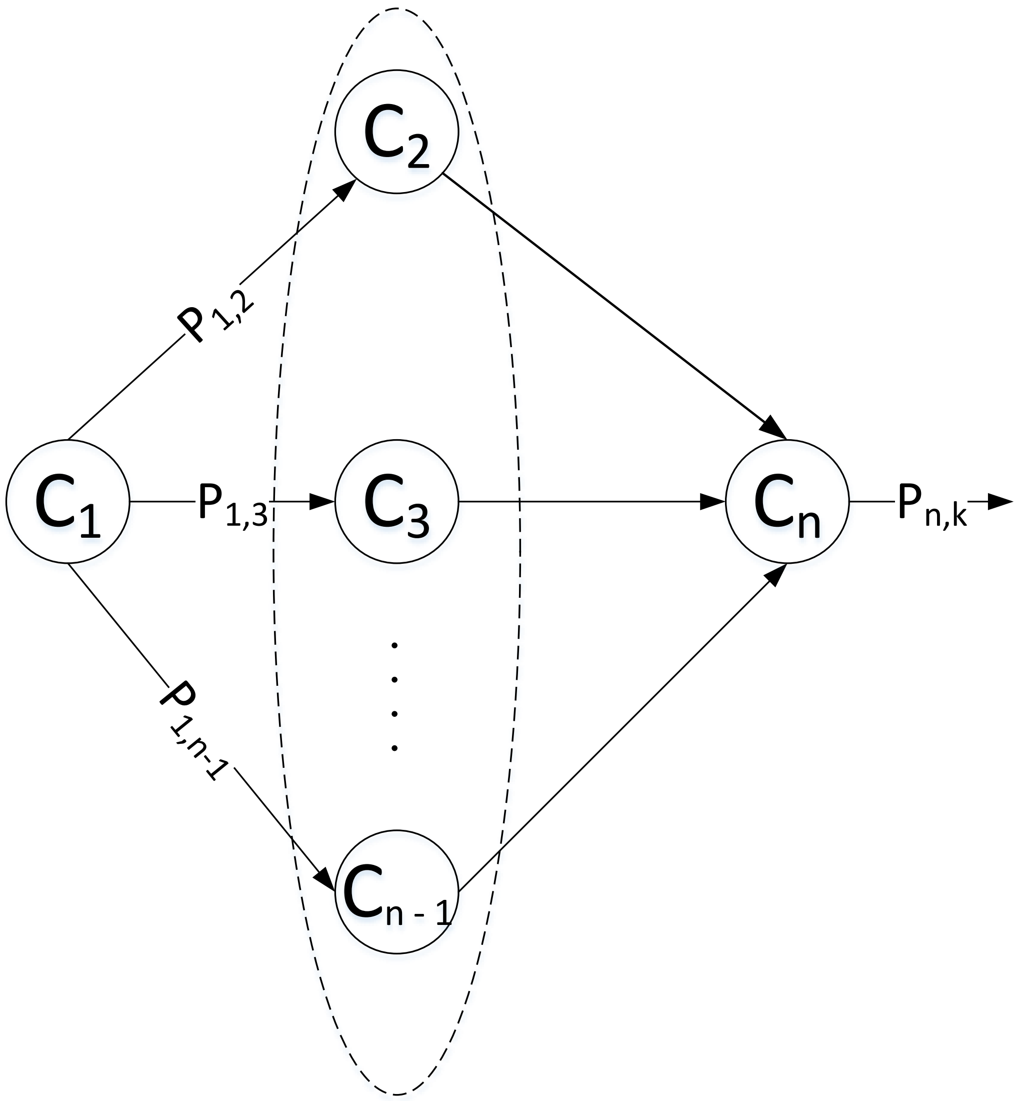

Figure 8: Transition between components in a parallel architecture

$$\begin{array}{c|cccccc}
 & \mathbf{C_1} & \mathbf{C_2} & \mathbf{C_3} & \cdots\cdots & \mathbf{C_{n-1}} & \mathbf{C_n} \\
\mathbf{C_1} & - & P_{1,2} & P_{1,3} & \cdots\cdots & P_{1,n-1} & - \\
\mathbf{C_2} & - & - & - & \cdots\cdots & & P_{2,n} \\
\mathbf{C_3} & - & - & - & \cdots\cdots & & P_{3,n} \\
 & . & . & . & \cdots\cdots & . & . \\
. & . & . & . & . & . & . \\
. & . & . & . & . & . & . \\
 & . & - & - & . & - & \\
\mathbf{C_{n-1}} & - & - & - & \cdots\cdots & - & P_{n-1,n} \\
\mathbf{C_n} & - & - & - & \cdots\cdots & - & -
\end{array}$$

Where $P_{1,2} = P_{1,3} = P_{1,n-1} = P_c$ and $P_{2,n} = P_{3,n} = P_{n-1,n} = P_k$

Figure 9: Transition Probability Matrix for the Parallel Structure.

$$\begin{array}{c|ccc}
 & \mathbf{C_1} & \mathbf{C_p} & \mathbf{C_n} \\
\mathbf{C_1} & - & P_c & - \\
\mathbf{C_p} & - & - & P_k \\
\mathbf{C_n} & - & - & -
\end{array}$$

Figure 10: Transformed transition probability matrix.

Evaluating the Transition Matrix of the Fault Tolerance Architectural Style

In this architecture, one component functions as primary and the others as backup if the primary component fails, as shown Figure 11. This means that if C_2 fails, C_3 acts as backup, and if C_3 fails, the next component within the demarcation acts as backup and so on up to C_{n-3}. Therefore, only one component at a time will function within this architecture. However, there is some level of parallelism which exists in this architecture as shown by the demarcation.

As a result, the components in the fault tolerance architecture can also be grouped collectively with the assumption that they all have the same transition probability. The transition probability matrix of the architecture (Figure 11) shown in Figure 12 can be transformed to the transition probability matrix as shown in Figure 13. The component utilization and the reliability can then be evaluated by applying Eq. 7 and Eq. 9 respectively.

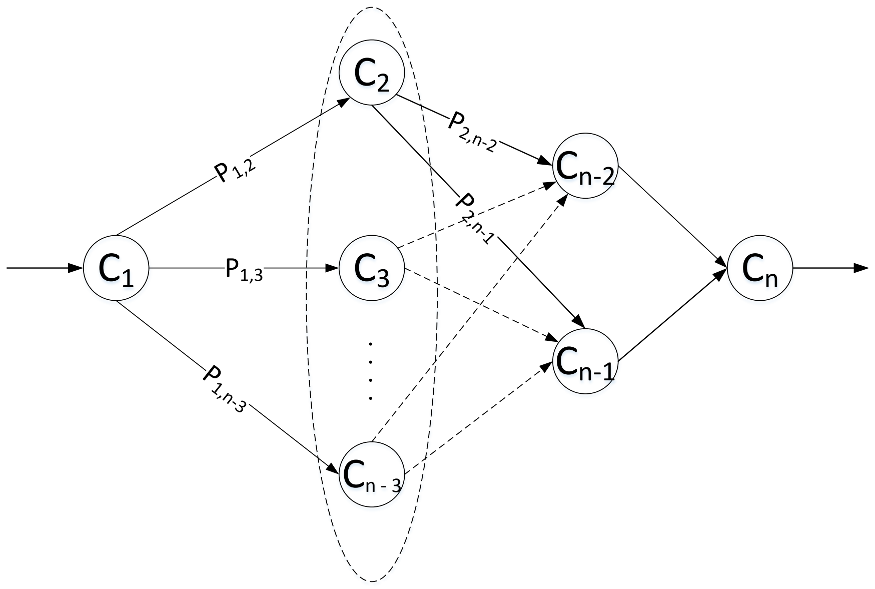

Figure 11: Fault tolerance architecture.

$$\begin{array}{l|llllllll}
 & \mathbf{C_1} & \mathbf{C_2} & \mathbf{C_3} & \cdots\cdots & \mathbf{C_{n-3}} & \mathbf{C_{n-2}} & \mathbf{C_{n-1}} & \mathbf{C_n} \\
\hline
\mathbf{C_1} & \text{-} & P_{1,2} & P_{1,2} & \cdots\cdots & P_{1,2} & \text{-} & \text{-} & \text{-} \\
\mathbf{C_2} & \text{-} & \text{-} & \text{-} & \cdots\cdots & \text{-} & P_{2,n-2} & P_{2,n-1} & \text{-} \\
\mathbf{C_3} & \text{-} & \text{-} & \text{-} & \cdots\cdots & & P_{2,n-2} & P_{2,n-1} & \text{-} \\
 & . & . & . & \cdots\cdots & . & . & . & . \\
. & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . \\
 & . & \text{-} & \text{-} & . & \text{-} & & \text{-} & \\
\mathbf{C_{n-3}} & \text{-} & \text{-} & \text{-} & . & \text{-} & P_{2,n-2} & P_{2,n-1} & \\
\mathbf{C_{n-2}} & \text{-} & \text{-} & \text{-} & \text{-} & \text{-} & \text{-} & \text{-} & P_{n-2,\, n} \\
\mathbf{C_{n-1}} & \text{-} & \text{-} & \text{-} & \cdots\cdots & \text{-} & \text{-} & \text{-} & P_{n-1,\, n} \\
\mathbf{C_n} & \text{-} & \text{-} & \text{-} & \cdots\cdots & \text{-} & \text{-} & \text{-} & \text{-}
\end{array}$$

Figure 12: Transition probability matrix for the fault tolerance structure.

$$\begin{array}{l|lllll}
 & \mathbf{C_1} & \mathbf{C_f} & \mathbf{C_{n-2}} & \mathbf{C_{n-1}} & \mathbf{C_n} \\
\hline
\mathbf{C_1} & \text{-} & P_{1,2} & \text{-} & \text{-} & \text{-} \\
\mathbf{C_f} & \text{-} & \text{-} & P_{2,n-2} & P_{2,n-1} & \text{-} \\
\mathbf{C_{n-2}} & \text{-} & \text{-} & \text{-} & \text{-} & P_{n-2,\, n} \\
\mathbf{C_{n-1}} & \text{-} & \text{-} & \text{-} & \text{-} & P_{n-1,\, n} \\
\mathbf{C_n} & \text{-} & \text{-} & \text{-} & \text{-} & \text{-}
\end{array}$$

Figure 13: Transformed transition probability matrix of the fault tolerance architecture.

Evaluating the Transition Probability for the Call-and-Return Architectural Style

In the call-and-return architecture, C_1 calls C_2 and C_2 returns control to C_1 as shown in Figure 14. The transition probability in this architecture is similar to that of the batch-sequential architecture where no transformation of the transition probability matrix is required. Therefore, the transition probability within a row of the matrix is stochastic.

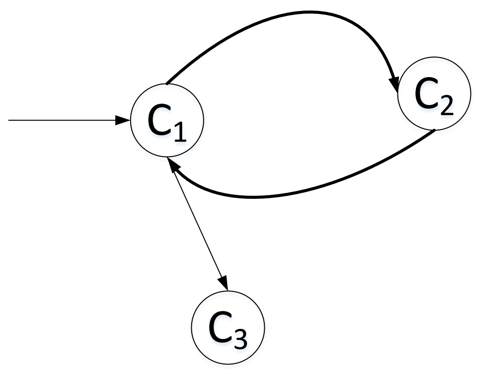

Figure 14: Call-and-return architecture.

Determining the Overall Software Reliability Based on the Hierarchical Framework

The requirements and steps for computing the overall reliability of a heterogeneous software system using the hierarchical framework are detailed as follows:

Input: Number of components, *n*, in the software.

Transition probability, $P_{i,j}$, from Component C_i to Component C_j*j*.

Reliability of component C_i, $R_{i,}$

Case for different architectural styles: Case 1 – batch-sequential or call and return, Case 2 – parallel and Case 3 – fault tolerance.

Output: The overall reliability, R, of the heterogeneous software architecture.

The steps to finding the overall reliability are as follows:

1. Identify the architectural styles in a system, based on the design specification of a system.
2. Develop the transition probability matrix based on the transition probability between components of the software.

3. Transform the transition matrix based on the architectural styles identified. If the transition probability is within a batch sequential or call-and-return style, the stochastic characteristics are not affected. However, if the style is parallel or fault-tolerance architecture, then the transition probability of components in this architecture should be grouped into a single entity to maintain the matrix stochastic characteristics.
4. Compute the component utilization or the expected number of visits per component.
5. Apply the value of the expected number of visits per component to the respective component reliability to compute the new component reliability based on component utilization.
6. Compute the overall reliability of the system.

Description of Case Studies

Four case studies were chosen to explore the application of the proposed hierarchical framework based on the Wang, Wu, and Chen (1999) and Gokhale and Trivedi (2002) model. Two cases are empirical structure of the software architecture and the other two are real systems which were used to test the accuracy of the reliability obtained for heterogeneous software architecture.

The case of the two empirical structures of the architecture and real systems was chosen from the study conducted by Wang, Wu, and Chen (1999) and Si, Yang, Wang, Huang, and Kavs (2010). The case studies provided the data needed for this study which included:

1. The structure of the software architecture
2. The functions of each component

3. Transition probabilities of the components
4. The reliability of each component.

Having this information from the selected case studies, the need to extract the structure of the software architecture, and determine the transition probabilities and reliability of each component was not necessary.

The four cases that will be analyzed are as follow:

1. Case 1 is an empirical structure of the software architecture used by Wang, Wu, and Chen (1999) to test the application of the model.
2. Case 2 is an empirical structure of the software architecture used by Si et al. (2010) to test the application of the model.
3. Case 3 is the architecture of the simulator of an ATM bank.
4. Case 4 is the architecture of the example of a Stock Trading System.

Case 1: Description of the Empirical Structure of the Software Architecture

Case 1 represents an empirical structure of a software architecture that has all the architectural styles described by Wang, Wu, and Chen (1999). The details of the software architecture, transition , and component reliabilities of Case 1 are shown in Figure 15, Table 5, and Table 6, respectively.

Case 1 architecture has 15 components where Component 1, C_1, represents the start state and component 15, C_{15}, the end state. The parallel structure within the architecture is represented by C_3 and C_4. This means that both components are executing simultaneously. C_5, C_8, and C_{11} represent the call-and-return style and C_{10} represents the backup of C_9. C_{11} and C_{12} transfer control back to C_1 and C_2 respectively to form a cyclic loop. The other components are running in a sequential manner where one executes followed by the other.

Table 5

Transition Probabilities of Case 1

$p_{1,2} = 0.40$	$p_{1,3} = p_{1,4} = 0.60$
$p_{2,6} = p_{3,7} = p_{4,7} = p_{5,8} = 1.00$	
$p_{6,7} = 0.10$	$p_{6,9} = p_{6,10} = 0.90$
$p_{7,11} = 0.25$	$p_{7,9} = p_{7,10} = 0.75$
$p_{8,5} = 0.20$	$p_{8,11} = 0.80$
$p_{9,12} = p_{10,12} = 0.70$	$p_{9,13} = p_{10,13} = 0.30$
$p_{11,1} = 0.15$	$p_{11,8} = 0.20$
$p_{11,13} = 0.50$	$p_{11,14} = 0.15$
$p_{12,2} = p_{12,13} = 0.50$	
$p_{13,14} = 0.40$	$p_{13,15} = 0.60$
$p_{14,15} = 1.00$	

Table 6

Component Reliabilities of Case 1

Component	**Reliability**
1	0.998
2	0.990
3	0.980
4	0.995
5	0.999
6	0.985
7	0.996
8	0.975
9	0.990
10	0.998
11	0.950
12	0.965
13	0.970
14	0.980
15	0.992

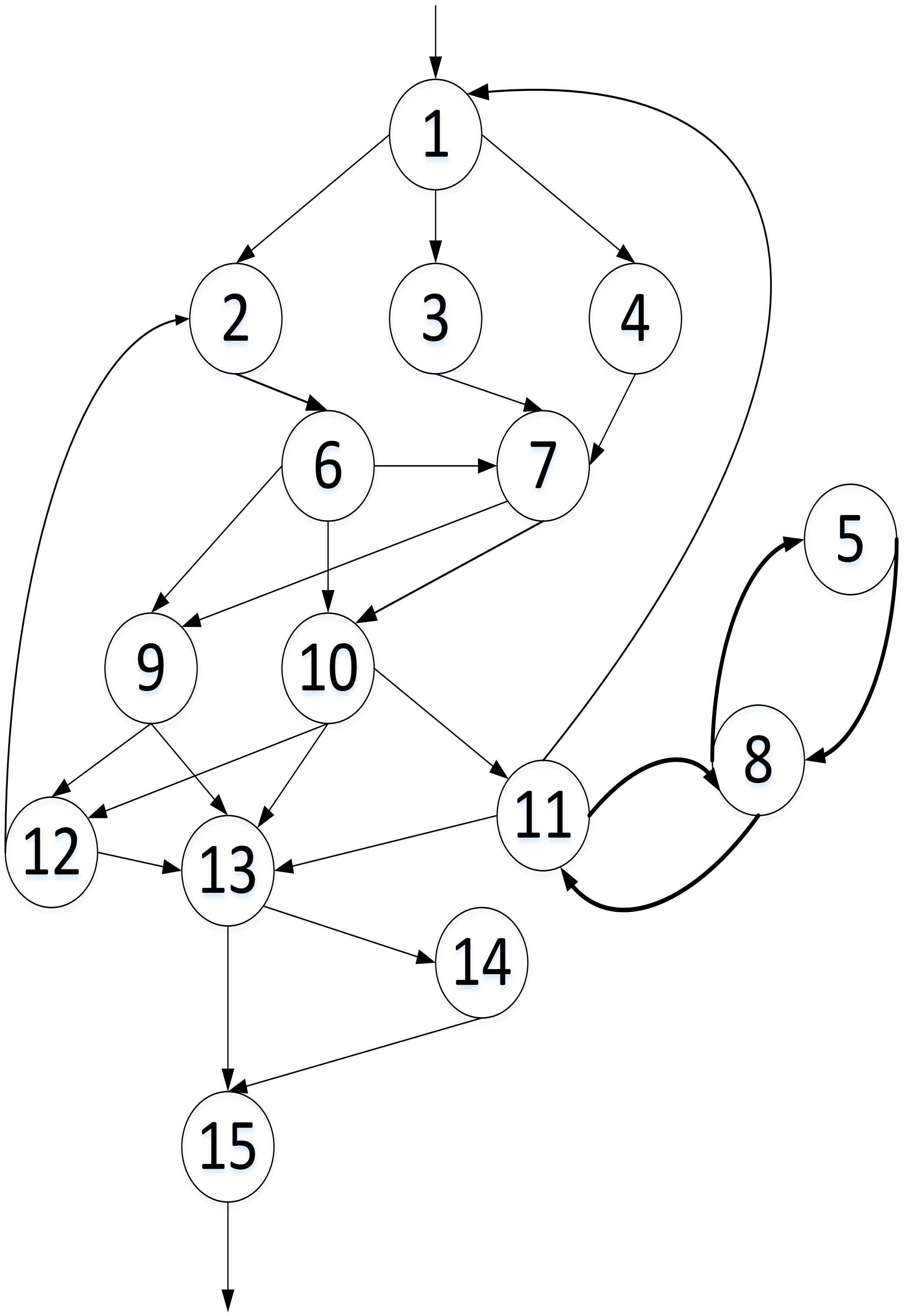

Figure 15: Software architecture of Case 1.

Case 2: Description of the Empirical Structure of the Software Architecture

Case 2 represents another empirical structure of a software architecture that has all the architectural styles described by Wang, Wu, and Chen (1999). The details of the software architecture, transition probabilities, and component reliabilities of Case 2 are shown in Figure 16, Table 7, and Table 8, respectively.

Case 2 architecture has 10 components where Component 1, C_1, represents the start state and component 10, C_{10}, the end state. The parallel structure within the architecture is represented by C_3 and C_4. This means that these components are executing simultaneously. C_6 and C_7 are in a fault-tolerance structure where C_6 is the primary component. C_8, and C_9 represent the call-and-return style C_5 transfers control back to C_1 and forms a cyclic loop. The other components are running in a sequential manner where one executes followed by the other.

Table 7

Transition Probabilities of Case 2

$p_{1,2} = 0.40$	$p_{1,3} = p_{1,4} = 0.60$
$p_{2,6} = p_{2,7} = 1.00$	
$P_{3,5} = 1.00$	
$P_{4,5} = 1.00$	
$P_{5,1} = 0.20$	$P_{5,10} = 0.80$
$P_{6,5} = 0.70$	$P_{6,8} = 0.30$
$P_{7,5} = 0.70$	$P_{7,8} = 0.30$
$P_{8,9} = 0.60$	$P_{8,10} = 0.40$
$P_{9,8} = 1.00$	

Table 8

Component Reliabilities of Case 2

Component	**Reliability**
1	0.975
2	0.990
3	0.985
4	0.987
5	0.970
6	0.999
7	0.985
8	0.992
9	0.985
10	0.994

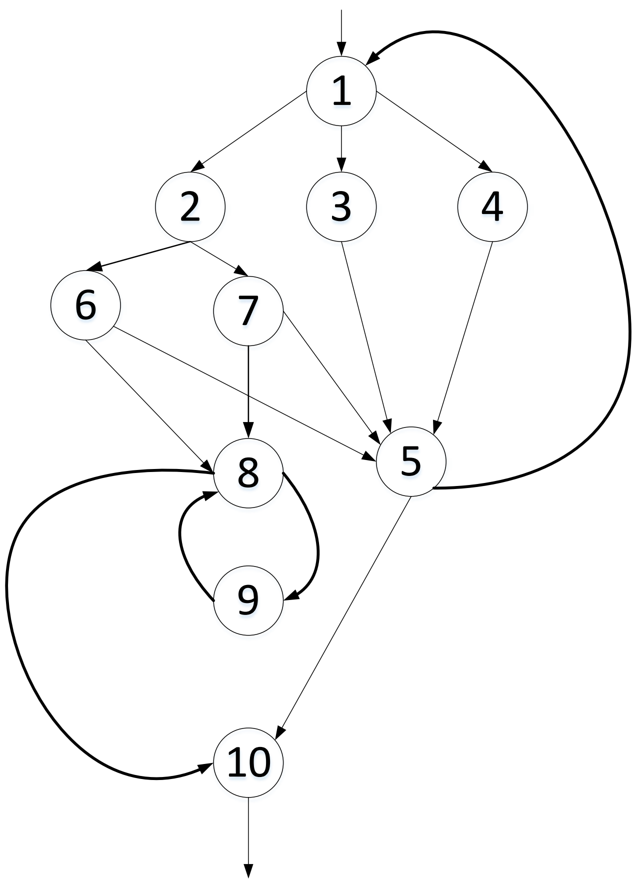

Figure 16: Software architecture of Case 2.

Case 3: Description of the Stock Trading System

This software is comprised of 11 components, and the transition probabilities, component reliabilities, and the software architecture are shown in Table 9, Table 10, and Figure 17 respectively. Failure data have been collected by this software for almost 3 years and were used to estimate the actual reliability of the system, which the results of the model were compared with. The components are listed as follows:

1. C_1 – represents the start component where the software is initiated.
2. C_2 – is the Trader.
3. C_3 – is the Exchange.
4. C_4 – is the Dispatcher.
5. C_5 – is the Dispatcher.
6. C_6 – is the Stock Manager.
7. C_7 – is the Stock Manager.
8. C_8 – is the Stock Manager.
9. C_9 – is the Printer.
10. C_{10} – is the Price Manager.
11. C_{11} – is the End state.

The Trader and Exchange make request to the Dispatcher, which is responsible for dispatching the request to the Stock Manager. The Stock Manager manages all the operations required during the trading session. The Price Manager provides the current price to the Stock Manager. The Transactor is responsible for processing the transactions, after which the information on trading is printed by the Printing component.

The parallel architectural style is represented by the three Stock Managers, C_6, C_7, and C_8, which are executing simultaneously. There are two Dispatchers, C_4 and C_5, where C_4 is the primary and C_5 is its backup. The other components execute in a sequential manner.

Table 9

Transition Probabilities of Case 4

$p_{1,2} = 0.489$	$p_{1,3} = 0.511$
$p_{2,4} = p_{2,5} = 1$	
$P_{3,4} = P_{3,5} = 1$	
$P_{4,6} = P_{4,7} = P_{4,8} = 0.333$	
$P_{5,6} = P_{5,7} = P_{5,8} = 0.3333$	
$P_{6,9} = 0.7$	$P_{6,10} = 0.3$
$P_{7,9} = 0.7$	$P_{7,10} = 0.3$
$P_{8,9} = 0.7$	$P_{8,10} = 0.3$
$P_{9,11} = 1$	
$P_{10,6} = P_{10,7} = P_{10,8} = 0.3333$	

Table 10

Component Reliabilities of Case 3

Component	**Reliability**
1	1.0
2	0.974
3	0.970
4	0.982
5	0.960
6	0.999
7	0.999
8	0.999
9	0.975
10	0.964
11	1.0

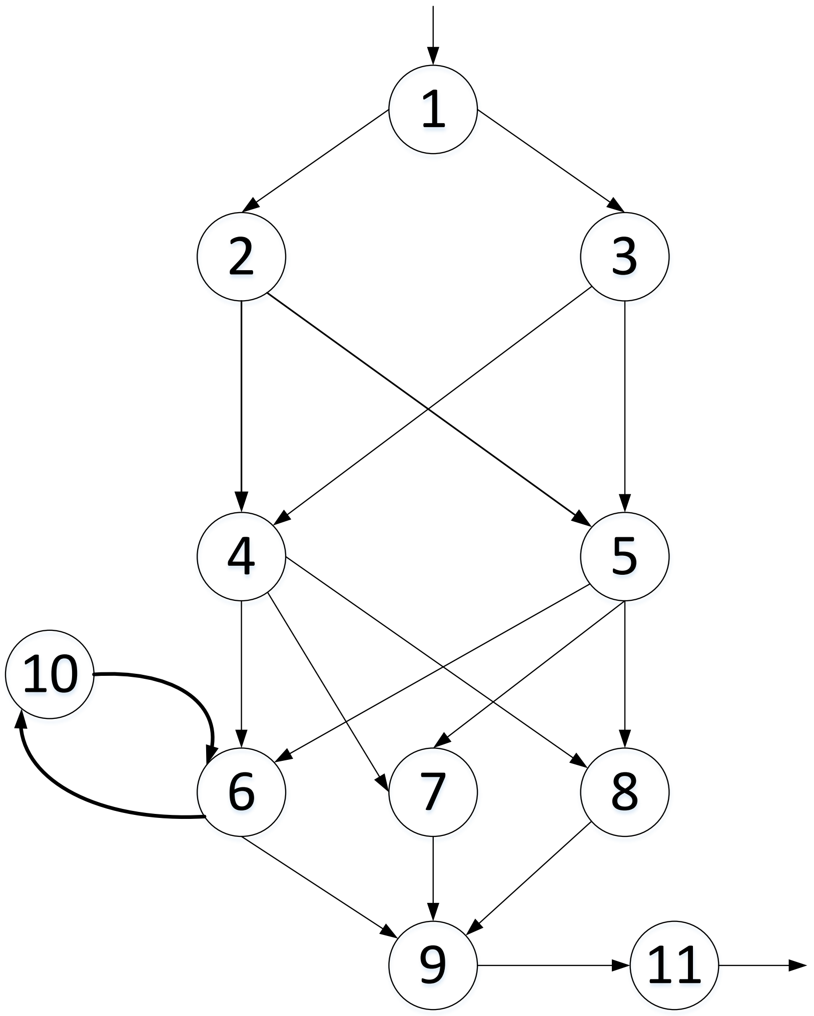

Figure 17: Software architecture of Case 3: Stock trading system.

Case 4: Description of the ATM Bank System Simulator

This software is comprised of 11 components, and the reliability of each component was found. The software has seven versions of which the overall reliability of the final version (version 7) was used as the actual reliability. The transition probabilities, component reliabilities, and the software architecture are shown in Table 11, Table 12, and Figure 18, respectively. The function of each component is listed as follows:

1. C_1 – represents the start component where the software is initiated.
2. C_2 – is the graphic user interface (GUI).

3. C_3 is the primary Database Management System (DBMS1).
4. C_4 – is the backup Database Management System (DBMS2) to C_3.
5. C_5 – is the identifier.
6. C_6 – is the Account Manager.
7. C_7 – is the Helper.
8. C_8 – is the Messenger.
9. C_9 – is the Transactor.
10. C_{10} – is the Verifier.
11. C_{11} – represents the end state.

C_3 and C_4 follow a fault tolerance style where C_4 acts as backup of C_3. C_6 and C_7 follow a call and return style where the account manager can call the helper multiple times before the account manager transfers control to other components.

Table 11

Transition Probabilities of Case 4

$p_{1,2} = 1.00$				
$p_{2,3} = p_{2,4} = 0.999$	$p_{2,11} = 0.001$			
$P_{3,5} = 0.227$	$P_{3,6} = 0.669$	$P_{3,8} = 0.104$		
$P_{4,5} = 0.227$	$P_{4,6} = 0.669$	$P_{4,8} = 0.104$		
$P_{5,2} = 0.048$	$P_{5,6} = 0.951$	$P_{5,11} = 0.001$		
$P_{6,3} = 0.4239$	$P_{6,4} = 0.4239$	$P_{6,7} = 0.1$	$P_{6,9} = 0.4149$	$P_{6,3} = 0.0612$
$P_{7,6} = P_{8,6} = 1.00$				

Table 12

Component Reliabilities of Case 4

Component	**Reliability**
1	1.0
2	0.982
3	0.97
4	0.96
5	1.0
6	0.996
7	0.99
8	1.0
9	1.0
10	0.8999
11	1.0

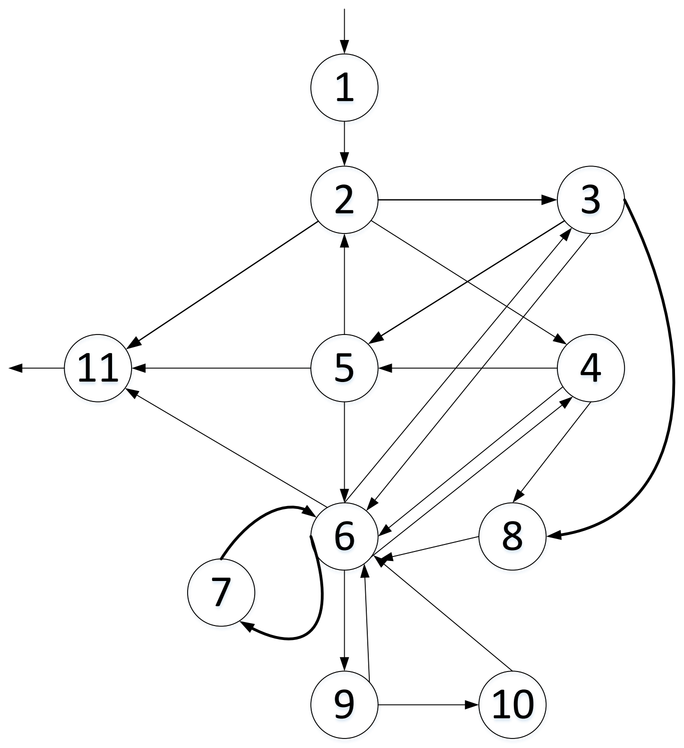

Figure 18: Software architecture of Case 4: ATM bank system simulator.

CHAPTER 4

RESULTS AND DISCUSSIONS

Transforming the Transition Probability Matrix for the Different Cases

The transition probability matrix and the transformed probability matrix for each case is shown in Tables 13 to 20.

Table 13

Transition Probability Matrix of Case 1

		Components														
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Components	1	0	0.4	0.6	0.6	0	0	0	0	0	0	0	0	0	0	0
	2	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
	3	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
	4	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
	5	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
	6	0	0	0	0	0	0	0.1	0	0.9	0.9	0	0	0	0	0
	7	0	0	0	0	0	0	0	0	0.75	0.75	0.25	0	0	0	0
	8	0	0	0	0	0.2	0	0	0	0	0	0.8	0	0	0	0
	9	0	0	0	0	0	0	0	0	0	0	0	0.7	0.3	0	0
	10	0	0	0	0	0	0	0	0	0	0	0	0.7	0.3	0	0
	11	0.15	0	0	0	0	0	0	0.2	0	0	0	0	0.5	0.15	0
	12	0	0.5	0	0	0	0	0	0	0	0	0	0	0.5	0	0
	13	0	0	0	0	0	0	0	0	0	0	0	0	0	0.4	0.6
	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
	15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 14

Transformed Transition Probability Matrix of Case 1

		Components												
		1	**2**	**C_p**	**5**	**6**	**7**	**8**	**C_f**	**11**	**12**	**13**	**14**	**15**
Components	**1**	0	0.4	0.6	0	0	0	0	0	0	0	0	0	0
	2	0	0	0	0	1	0	0	0	0	0	0	0	0
	C_p	0	0	0	0	0	1	0	0	0	0	0	0	0
	5	0	0	0	0	0	0	1	0	0	0	0	0	0
	6	0	0	0	0	0	0.1	0	0.9	0	0	0	0	0
	7	0	0	0	0	0	0	0	0.75	0.25	0	0	0	0
	8	0	0	0	0.2	0	0	0	0	0.8	0	0	0	0
	C_f	0	0	0	0	0	0	0	0	0	0.7	0.3	0	0
	11	0.15	0	0	0	0	0	0.2	0	0	0	0.5	0.15	0
	12	0	0.5	0	0	0	0	0	0	0	0	0.5	0	0
	13	0	0	0	0	0	0	0	0	0	0	0	0.4	0.6
	14	0	0	0	0	0	0	0	0	0	0	0	0	1
	15	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 15

Transition Probability Matrix of Case 2

		Components									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Components	**1**	0	0.4	0.6	0.6	0	0	0	0	0	0
	2	0	0	0	0	0	1	1	0	0	0
	3	0	0	0	0	1	0	0	0	0	0
	4	0	0	0	0	1	0	0	0	0	0
	5	0.2	0	0	0	0	0	0	0	0	0.8
	6	0	0	0	0	0.7	0	0	0.3	0	0
	7	0	0	0	0	0.7	0	0	0.3	0	0
	8	0	0	0	0	0	0	0	0	0.6	0.4
	9	0	0	0	0	0	0	0	1	0	0
	10	0	0	0	0	0	0	0	0	0	0

Table 16

Transformed Transition Probability Matrix of Case 2

Components	1	2	C_p	5	C_f	8	9	10
1	0	0.4	0.6	0	0	0	0	0
2	0	0	0	0	1	0	0	0
C_p	0	0	0	1	0	0	0	0
5	0.2	0	0	0	0	0	0	0.8
C_f	0	0	0	0.7	0	0.3	0	0
8	0	0	0	0	0	0	0.6	0.4
9	0	0	0	0	0	1	0	0
10	0	0	0	0	0	0	0	0

Table 17

Transition Probability Matrix of Case 3

Components	1	2	3	4	5	6	7	8	9	10	11
1	0	1	0	0	0	0	0	0	0	0	0
2	0	0	0.999	0.999	0	0	0	0	0	0	0.001
3	0	0	0	0	0.227	0.669	0	0.104	0	0	0
4	0	0	0	0	0.227	0.669	0	0.104	0	0	0
5	0	0.048	0	0	0	0.951	0	0.001	0	0	0
6	0	0	0.4239	0.4239	0	0	0.1	0	0.4149	0	0.0612
7	0	0	0	0	0	1	0	0	0	0	0
8	0	0	0	0	0	1	0	0	0	0	0
9	0	0	0	0	0	0.01	0	0	0	0.99	0
10	0	0	0	0	0	1	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0

Table 18

Transformed Transition Probability Matrix of Case 3

Components	1	2	C_f	5	6	7	8	9	10	11
1	0	1	0	0	0	0	0	0	0	0
2	0	0	0.999	0	0	0	0	0	0	0.001
C_f	0	0	0	0.227	0.669	0	0.104	0	0	0
5	0	0.048	0	0	0.951	0	0.001	0	0	0
6	0	0	0.4239	0	0	0.1	0	0.4149	0	0.0612
7	0	0	0	0	1	0	0	0	0	0
8	0	0	0	0	1	0	0	0	0	0
9	0	0	0	0	0.01	0	0	0	0.99	0
10	0	0	0	0	1	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0

Table 19

Transition Probability Matrix of Case 4

Components	1	2	3	4	5	6	7	8	9	10	11
1	0	0.489	0.511	0	0	0	0	0	0	0	0
2	0	0	0	1	1	0	0	0	0	0	0
3	0	0	0	1	1	0	0	0	0	0	0
4	0	0	0	0	0	0.33	0.33	0.34	0	0	0
5	0	0	0	0	0	0.33	0.33	0.34	0	0	0
6	0	0	0	0	0	0	0	0	0.7	0.3	0
7	0	0	0	0	0	0	0	0	0.7	0.3	0
8	0	0	0	0	0	0	0	0	0.7	0.3	0
9	0	0	0	0	0	0	0	0	0	0	1
10	0	0	0	0	0	0.33	0.33	0.34	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0

Table 20

Transformed Transition Probability Matrix of Case 4

Components		1	2	3	C_f	6	7	8	9	10	11
	1	0	0.489	0.511	0	0	0	0	0	0	0
	2	0	0	0	1	0	0	0	0	0	0
	3	0	0	0	1	0	0	0	0	0	0
	C_f	0	0	0	0	0.33	0.33	0.34	0	0	0
	6	0	0	0	0	0	0	0	0.7	0.3	0
	7	0	0	0	0	0	0	0	0.7	0.3	0
	8	0	0	0	0	0	0	0	0.7	0.3	0
	9	0	0	0	0	0	0	0	0	0	1
	10	0	0	0	0	0.33	0.33	0.34	0	0	0
	11	0	0	0	0	0	0	0	0	0	0

Analysis of Results

The results based on the hierarchical approach taken for estimating the overall system reliability for each case are shown in Table 21. It showed that the hierarchical framework could be used within reasonable accuracy for estimating reliability in heterogeneous architecture based on the results for Cases 1, 2, and 3, as the difference between the actual reliability and the reliability estimated was under 12%. The difference between the actual reliability and that of the hierarchical framework that was used in this study for Cases 2 and 3, in particular, were 5.12% and 0.82%. Figure 19 graphically shows the extent to which the actual reliability was in agreement with the reliability obtained by the proposed hierarchical framework.

Table 21

The Overall System Reliability for each Case compared to the Actual Reliability

Case	System Reliability based on proposed approach	Actual System Reliability	% Difference
Case 1	0.83317	0.878183	5.12%
Case 2	0.893038	0.8039	11.09%
Case 3	0.872053	0.865	0.82%
Case 4	0.251755	0.526	-52.14%

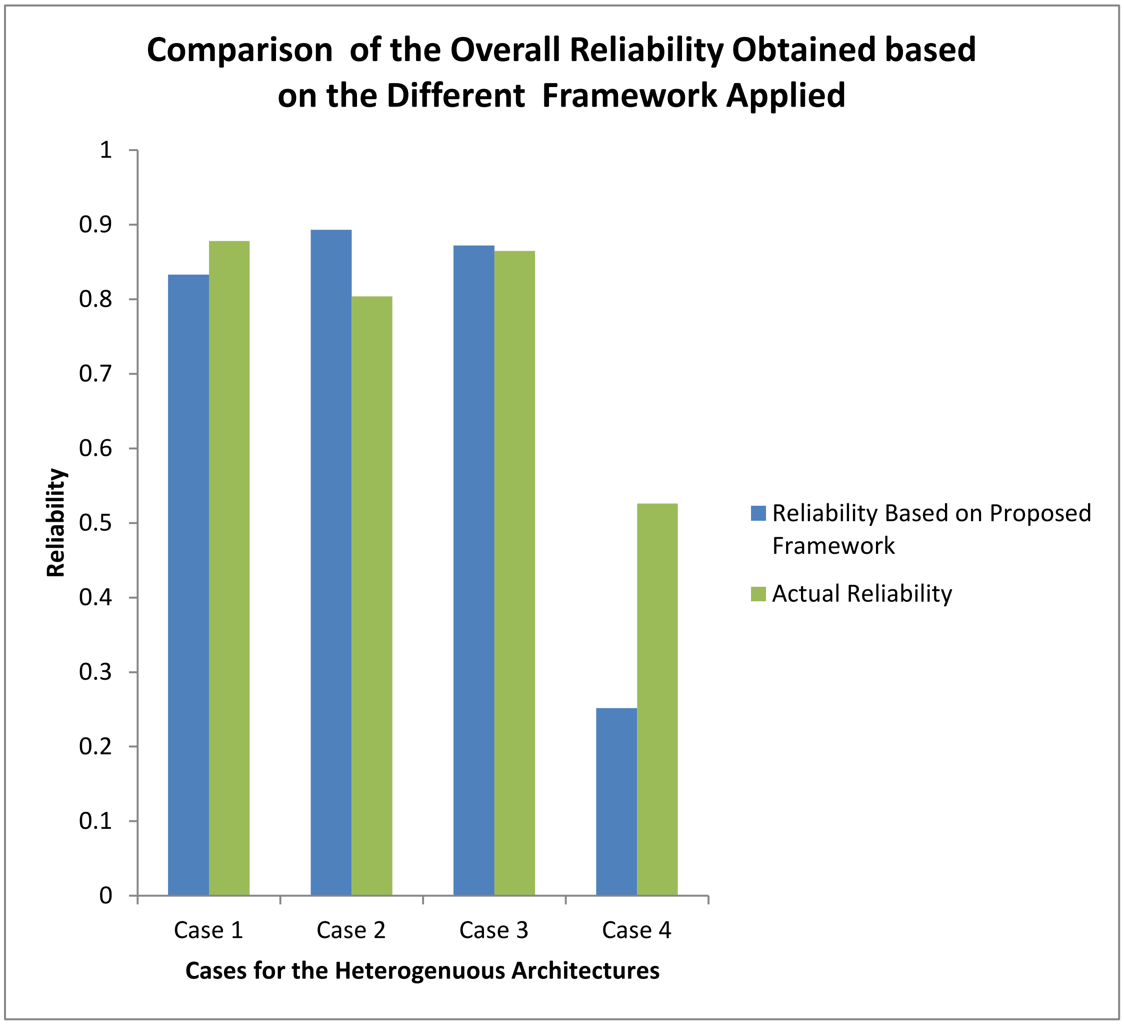

Figure 19: Comparison of the reliability obtained based on the proposed framework with the actual reliability.

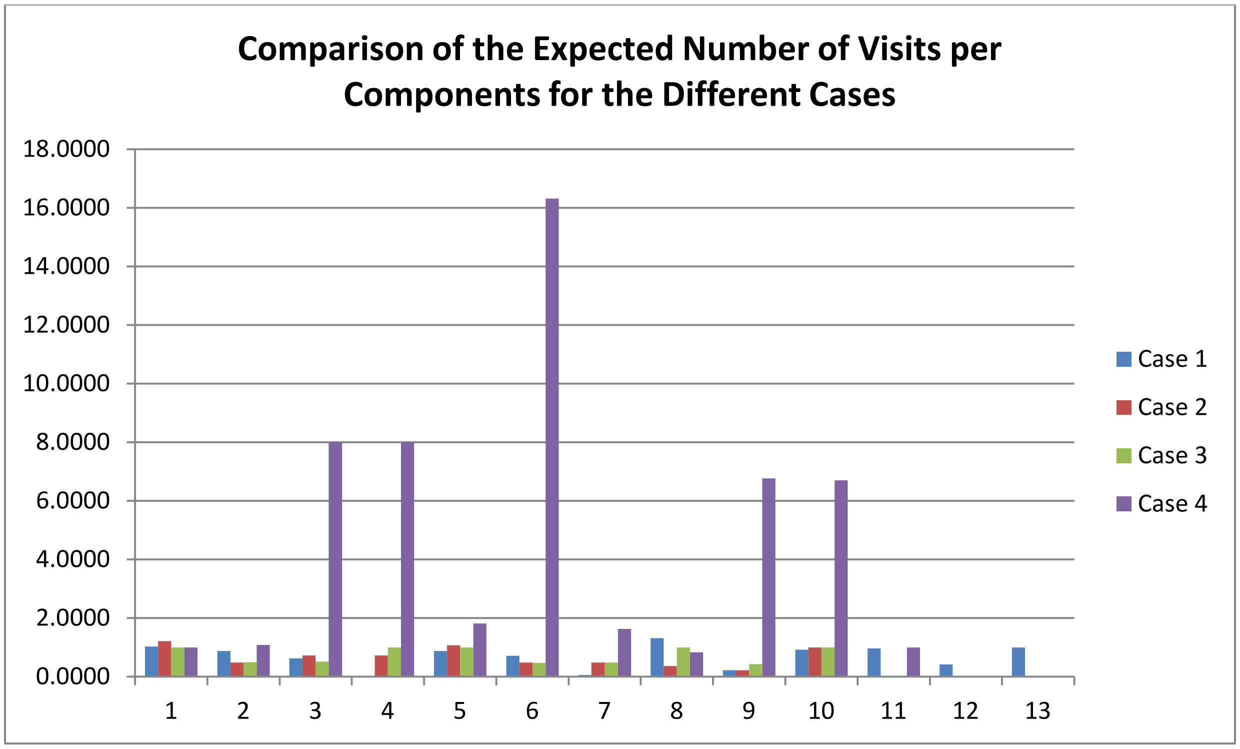

Figure 20: Comparison of the expected number of visits per components for the different cases.

The hierarchical approach, however, was not valid for estimating the overall reliability of the architecture of Case 4 as the reliability estimated was not in agreement with the actual reliability of the system. This was most likely due to the more frequent interactions or utilization of the components in Case 4 architecture as shown in Figure 20.

The number of visits per component in Case 4 is much higher when compared to the other cases and therefore resulted in much lower reliability estimation.

The model for calculating reliability relies on the expected number of visits as an input parameter, and thus a greater value for the expected number of visits would result in a lower reliability. This is true from a practical perspective of software components that the more frequent a component is utilized, the greater the likelihood for failure. This shows that the hierarchical approach has its limitations for estimating reliability depending on the architectural styles, the utilization frequency of components, and the complexity of how components interact with each other.

Given the assumptions that were made for evaluating heterogeneous architecture, the model does not purely represent the consistency or extent to which software fails and therefore introduces a certain level of uncertainty when determining software reliability by using the Markov's process to capture the number of visits per component. Additional factors, such as the methodology used to extract the architectural information, would have to be considered when analyzing the heterogeneity of the software architecture for accurately and consistently using the expected number of visits as a parameter for determining the overall reliability of heterogeneous architectural styles.

While the hierarchical approach has its limitations in respect of the complexity of the software architecture to which it can be applied, it could be very useful as a reliability indicator to identify how components will be expected to be utilized and which component most likely would require special resources to ensure that high system reliability is maintained. The accuracy of the hierarchical approach could be improved by using higher order Taylor series. In addition, this approach also could be extended to analyze or improve performance within a heterogeneous architectural setting where the parameter of time is introduced as the composite approach makes performance and sensitivity analysis intractable and thus the analysis of reliability and failure becomes generalized.

The results further support Koseva assessment that the hierarchical approach does not accurately estimate software reliability for every system. However, this study went further by critically exploring the usefulness of the hierarchical approach on heterogeneous architectural styles and, in the process, observed that the frequency of component interactions strongly influenced accuracy of the method, which was not highlighted as a major drawback in other research work. While accuracy is required at times for estimating reliability, most often data are not available and therefore very accurate estimation may not be possible even with models that are considered highly accurate. As a result, the hierarchical approach could be applicable to provide reasonable reliability estimation in the software design stage and used to critically provide insights on failure and time dependency. With the inclusion of failure behavior and time dependency in the estimation of reliability for heterogeneous architecture, mean time to failure would also be incorporated as another parameter that coincides with failure behavior and provides added tractability for reliability, sensitivity, and performance analysis.

The Limitations of the Hierarchical Approach and the Proposed Hierarchical Framework

Most studies had focused on systems that function in a sequential manner and did not explore the limitations of the hierarchical approach for complex systems with different architectural structures, which is what was revealed from the results obtained from the cases that were analyzed. The hierarchical approach tends to be much more accurate when applied to estimating reliability for sequential systems or where components are not frequently utilized. This can be observed in Case 3 where components (Figure 23), though having heterogeneous architectural styles, do not have much interaction among each other. In comparison to component 6 of Case 4 (Figure 20), which has the highest number of visits, it was observed that this component was utilized most and thus had a much greater tendency to fail. The utilization of components in Cases 1, 2 and 3 (Figure 21, Figure 22 and Figure 23) was less than one and thus did not substantially affect the accuracy of the reliability estimation. However, the opposite was true for Case 4 (Figure 24).

With high component utilization, the proposed hierarchical approach becomes less accurate and would require other approaches for estimating reliability more accurately. For systems that do not have a great level of complexity where components interaction is not very frequent, the hierarchical approach can be accurate and reasonably useful in estimating reliability for heterogeneous architecture.

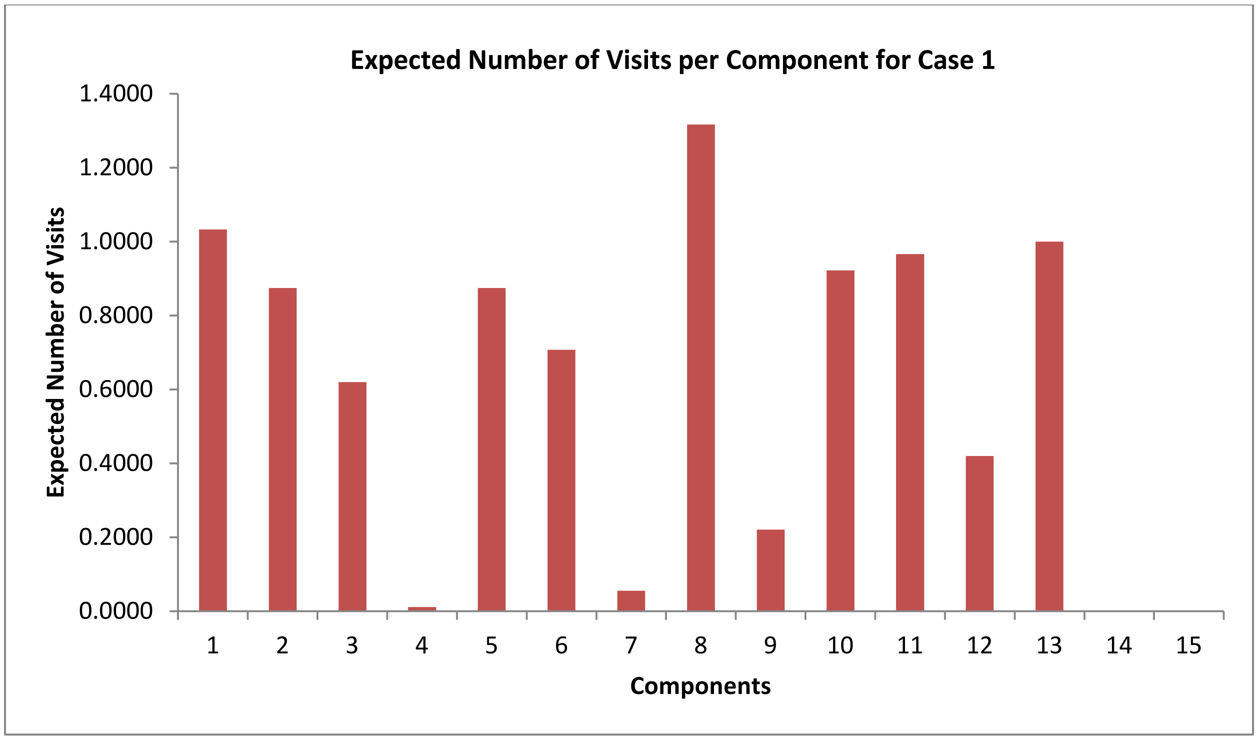

Figure 21: Expected number of visits per component for Case 1 architecture.

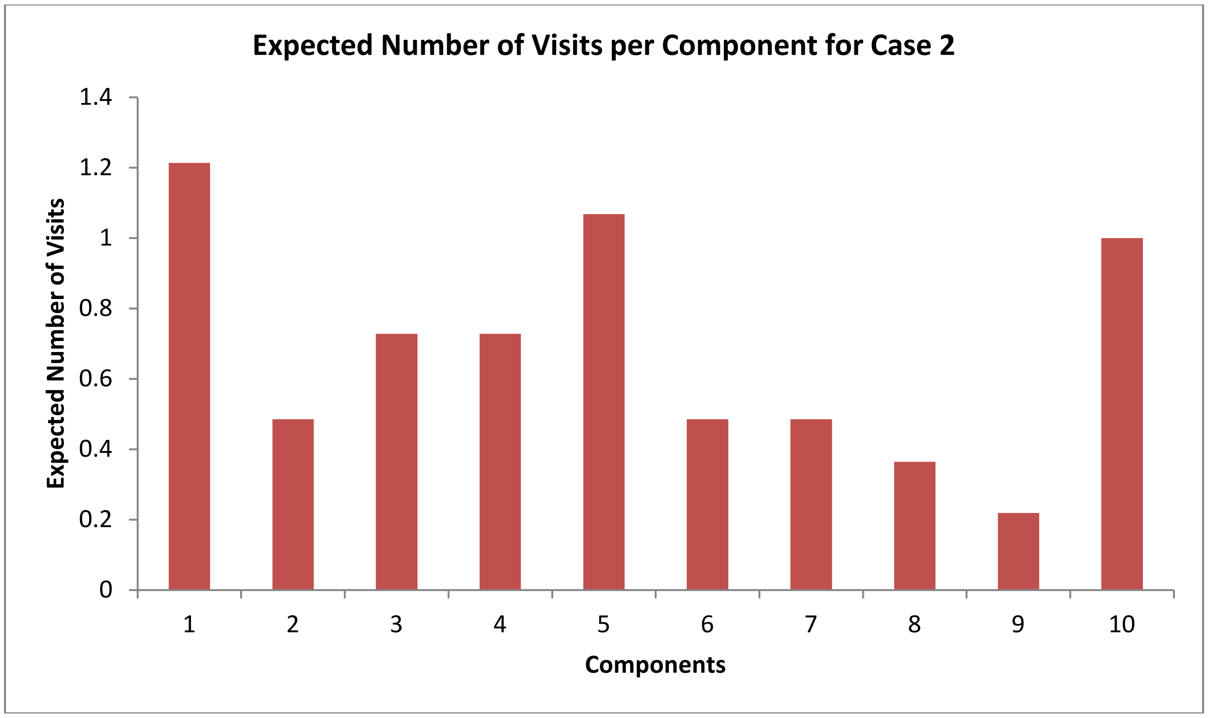

Figure 22: Expected number of visits per component for Case 2 architecture.

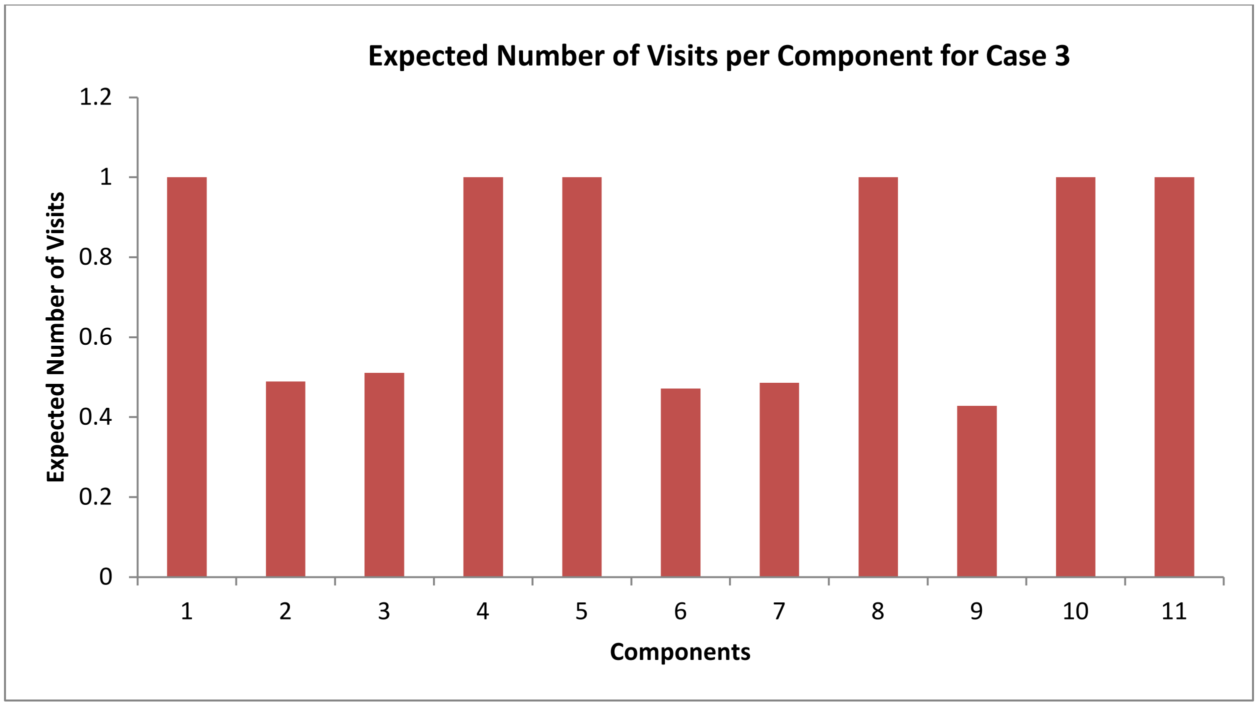

Figure 23: Expected number of visits per component for Case 3 architecture.

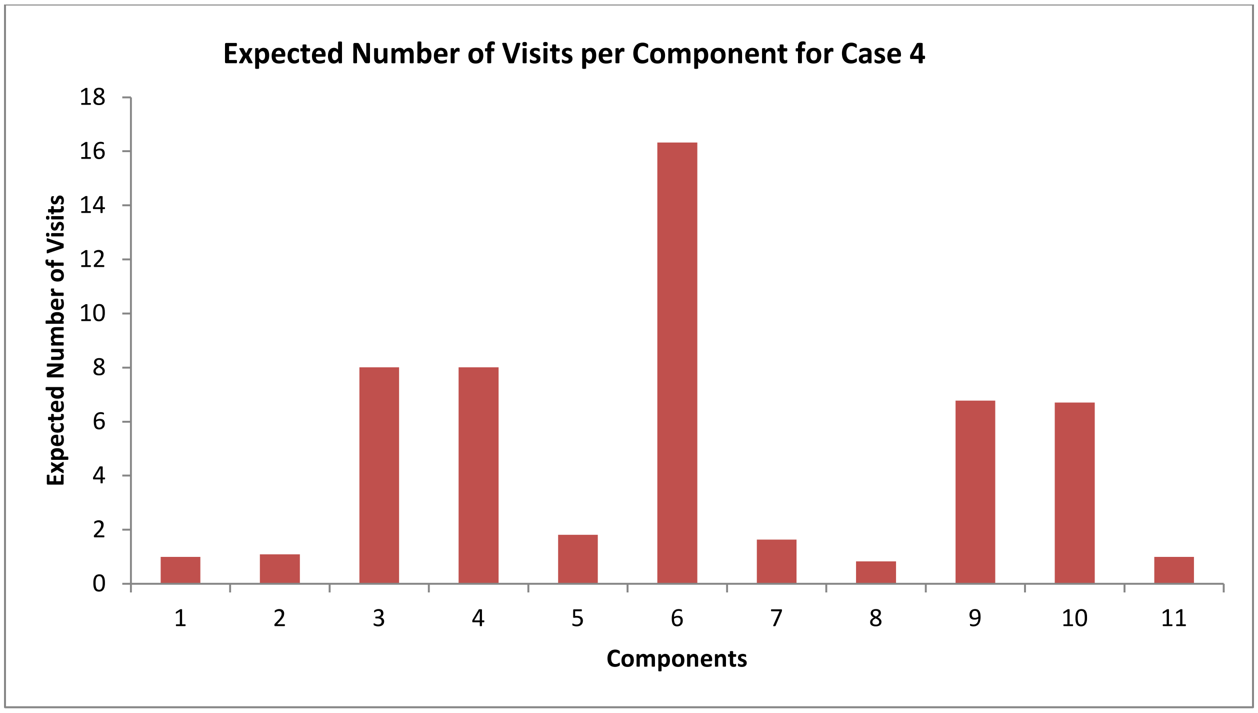

Figure 24: Expected number of visits per component for Case 4 architecture.

CHAPTER 5

CONCLUSION

An exploration of the use of a hierarchical framework to estimate reliability for heterogeneous software architecture was conducted. A hierarchical approach for estimating software reliability becomes relevant and useful due to its ability to offer more flexibility and in-depth reliability analysis for improving its method of solution than does the composite model approach.

Very few studies have been done on finding reliability models for heterogeneous software architecture, and none, to my knowledge, have been done on using the hierarchical approach for estimating reliability for heterogeneous software architecture. As a result, my main purpose and contributions from this study were to present a simpler, accurate alternative approach while observing the effectiveness of the proposed hierarchical framework for reliability estimation of heterogeneous software architecture.

The proposed hierarchical framework was developed based on the concepts proposed by Wang, Wu, and Chen (1999) and Gokhale and Trivedi (2002). Both models are state-based models and assumed an absorbing DTMC process. This means that there is a start and end state (absorbing states), and that components to be executed in the next state will only depend on components of the current state and the component of the next state will not have any dependency to the past history of the current state.

Wang, Wu, and Chen (1999) follow a composite approach to estimating heterogeneous software reliability and identified four software architectural styles: batch-sequential, parallel filter, fault tolerance, and call-and-return. Gokhale and Trivedi (2002) uses a hierarchical approach based on component reliability and component utilization or expected number of visits per component but had been applied only to a system that functions in a sequential manner.

The approach to developing the proposed hierarchical framework involved: the identification of the architectural styles, development of the transition probability matrix, transformation of the transition matrix based on the architectural styles identified, finding the component utilization and applying the Gokhale and Trivedi (2002) model to compute overall reliability.

To test the proposed hierarchical framework of this study, four case studies were taken from research conducted by Wang, Wu, and Chen (1999) and Si et al. (2010) as the basis for comparison. It was assumed that the data from the case studies were an accurate representation of the software systems that were used.

Generally, the proposed hierarchical framework was comparable to the actual reliability of the software systems used in the case studies with the exception of Case 4 where results were in total disagreement. This was due most likely to the much higher interactions of the components in Case 4 architecture as the components utilization was much higher when compared to other cases. However, based on the results, the proposed hierarchical framework of this study is more accurate and useful in software systems that do not have very high interactions among its components. This was, to my knowledge, a newly discovered observation that has not been expressed by other studies using the hierarchical approach. While there are limitations in the proposed hierarchical framework of this study, it still can be used as a reliability indicator when very high accuracy is not desired.

Future Work

The use of a hierarchical approach for estimating reliability of heterogeneous software architecture has great potential and there is much that is left to be explored. One area that would be the focus of future interest is to apply this framework to other software systems to validate the range to which the proposed hierarchical framework can be applied. With this information, it is possible to state with absolute certainty the extent of the application of the framework which would better improve its effectiveness in areas where it might be needed. In addition, quantification of the impact of component utilization and transition probabilities on overall reliability based on the different architectural styles could also be determined. This would introduce innovative ways for conducting sensitivity analysis on heterogeneous component systems so that more light could be shed on discovering more information on how reliability is affected and subsequently its improvement.

Exploring a CTMC or SMP approach to estimating reliability of heterogeneous software architecture is also worth investigating. In addition to the benefits of finding new ways to perform sensitivity analysis, the CTMC or SMP goes further by including performance analysis. As a result, the improvement of these areas will not only improve reliability but also improve software efficiency and performance as well.

APPENDIX

DETAILS OF RESULTS

Table 22

Component Reliability, Expected Number of Visits and Overall Reliability from the Proposed Hierarchical Framework Applied to Case 1

Case 1															
Component	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Reliability	0.998	0.99	0.98	0.995	0.999	0.985	0.996	0.975	0.99	0.998	0.95	0.965	0.97	0.98	0.992
$\boldsymbol{X_{i,j}}$	1.0332	0.8744	0.6199	0.0111	0.8744	0.7073	0.0553	1.3174	0.2210	0.9222	0.9668	0.4199	1.00		
$\mathbf{R}^{Xi,j}$	0.9979	0.9913	0.9876	0.9969	0.9991	0.9894	0.9998	0.9672	0.9868	0.9974	0.9516	0.9852	0.97	1	1
$\prod_{i=1}^{n} R_i^{X_{i,j}}$	0.83317														

Table 23

Component Reliability, Expected Number of Visits and Overall Reliability from the Proposed Hierarchical Framework Applied to Case 2

Case 2										
Component	1	2	3	4	5	6	7	8	9	10
Reliability	0.975	0.99	0.985	0.987	0.97	0.99	0.985	0.992	0.985	0.994
$\boldsymbol{X_{i,j}}$	1.213592	0.485437	0.728155	0.728155	1.067961	0.485437	0.485437	0.364078	0.218447	1
$\mathbf{R}^{Xi,j}$	0.969742	0.995133	0.989055	0.990517	0.967994	0.995133	0.99269	0.99708	0.996704	0.994
$\prod_{i=1}^{n} R_i^{X_{i,j}}$	0.893038									

Table 24

Component Reliability, Expected Number of Visits and Overall Reliability from the Proposed Hierarchical Framework Applied to Case 3

Case 3											
Component	1	2	3	4	5	6	7	8	9	10	11
Reliability	1	0.974	0.97	0.982	0.96	0.999	0.999	0.999	0.975	0.964	1
$X_{i,j}$	1	0.489	0.511	1	1	0.471429	0.485714	1	0.428571	1	1
$\mathbf{R}^{Xi,j}$	1	0.9872	0.984556	0.982	0.96	0.999528	0.999514	0.999	0.989208	0.964	1
$\prod_{i=1}^{n} R_i^{X_{i,j}}$	0.872053										

Table 25

Component Reliability, Expected Number of Visits and Overall Reliability from the Proposed Hierarchical Framework Applied to Case 4

Case 4											
Component	1	2	3	4	5	6	7	8	9	10	11
Reliability	1	0.982	0.97	0.96	1	0.996	0.99	1	1	0.8999	1
$X_{i,j}$	1	1.087223	8.005076	8.005076	1.817152	16.3221	1.63221	0.834345	6.772041	6.704321	1
$\mathbf{R}^{Xi,j}$	1	0.980445	0.783622	0.72124	1	0.936675	0.98373	1	1	0.493064	1
$\prod_{i=1}^{n} R_i^{X_{i,j}}$	0.251755										

REFERENCE LIST

REFERENCE LIST

Chandran, S. K., Dimove, A., & Punnekkat, S. (2010). *Modeling uncertainties in the estimation of software reliability–A pragmatic approach*. Fourth IEEE International Conference on Secure Software Integration and Reliability Improvement, Singapore.

Cheung, R. (1980, March). A user-oriented software reliability model. *IEEE Transactions on Software Engineering, SE-6*.

Fair, W., & Smith, O. (1988). *Statistical modeling and estimation of reliability functions for software (SMERFS) user's guide* (1st ed.). Dahlgren,VA: NSWC.

Farr, W. (1996). Handbook of software reliability engineering. In M. Lyu (Ed.), *Software reliability modeling survey* (pp. 71-117). New York: McGraw-Hill.

Fenlason, J., & Stallman, R. (2013, September 17). *Gnu gprof*. Retrieved December 18, 2013, from GNU Operating System: http://www.gnu.org/doc/doc.html

Gokhale, S. (2002). *Accurate reliability prediction based on software structure*. Retrieved December 17, 2013, from http://www.engr.uconn.edu/~ssg/cse300/397-232.pdf

Gokhale, S., & Trivedi, K. (1997). Structure-based software reliability prediction. *Proceedings of the 5th International Conference of Advanced Computing (ADCOMP)*, 447-452.

Gokhale, S., & Trivedi, K. S. (2002). *Reliability prediction and sensitivity analysis based on software architecture*. Proceedings of the 13th International Symposium on Software Reliability Engineering (ISSRE'02), Washington, DC.

Goseva-Popstanova, K., Mathur, A., & Kishor, T. (2001). Comparison of software reliability models. *Software Reliability Engineering*, 22-31.

Goseva-Popstojanova, K., & Kamavaram, S. (2003). *Assessing uncertainty in reliability of component-based software systems*. International Symposium on Software Reliability Engineering, Morgantown, WV.

Goseva-Popstojanova, K., & Trivedi, K. S. (2001). Architecture-based approach to reliability assessment of software systems. *Performanc Evaluation: An International Journal*, 179-204.

Goseva-Popstojanova, K., Hamill, M., & Perugupalli, R. (2005). *Large empirical case study of architecture-based software reliability*. Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering (ISSRE'05), Washington, DC.

Goseva-Popstojanova, K., Trivedi, K., & Mathur, A. (2000). *How different architecture based software reliability models are related.* International Symposium on Software Reliability Engineering (ISSRE), San Jose, CA.

Grinstead, C., & Snell, J. L. (2006). *Introduction to probability* (2nd ed.). The American Mathematical Society. Retrieved November 19, 2013, from http://www.dartmouth.edu/~chance/teaching_aids/books_articles/probability_book/Chapter11.pdf

Hamlet, D. (1992). Are we testing for true reliability? *IEEE Software, 13*(4), 21-27.

Horgan, J. R., & Mathur, A. P. (1996). Software testing and reliability. In M. Lyu (Ed.), *Handbook of software reliability engineering* (pp. 531-566). New York: McGraw-Hill.

Koziolek, H., Schlich, B., & Bilich, C. (2010). A large-scale industrial case study on architecture-based software reliability analysis. *IEEE Computer Society*, 279-288.

Markov chain. (2014, February 9). In *Wikipedia, the free encyclopedia.* Retrieved February 14, 2014, from: http://en.wikipedia.org/wiki/Markov_chain

Michael, L. R. (1996). *Handbook of software reliability engineering.* New York: McGraw-Hill.

Miller, K., Morell, L., Noonan, R., Park, S., Nicol, D., Murrill, B., & Voas, J. (1992, January). Estimating the probability of failure when resting reveals no failures. *IEEE Transactions on Software Engineering, 18*(1), 33-42.

Nikora, A. (2002). *Computer-aided software reliability estimation user's guide (CASRE).* Pasadena, CA: Author.

Nikora, A., & Lyu, M. (1999). *Software reliability and risk management: Techniques and tools.* International Symposium on Software Reliability Engineering, Boca Raton, FL.

Parnas, D. (1975). *Influence of software structure on reliability*. Proceedings 1975 International Conference on Reliable Software, Los Angeles, CA.

Perugupalli, R. (2004). *Empirical assessment of architecture-based reliability of open-source software.* Morgantown, WV: West Virginia University.

Shaw, M. (1993). *Software architectures for shared information systems* (Technical Report). Carnegie Mellon University, School of Computer Science, Pittburgh, PA.

Si, Y., Yang, X., Wang, X., Huang, C., & Kavs, A. J. (2010). An architecture-based reliability estimation framework through component composition mechanisms. *2nd International Conference on Computer Engineering and Technology, 2*, 165-170.

Team, T. S. (1998). *χSuds software understanding system: User's manual*. Retrieved from https://www.cs.purdue.edu/homes/apm/foundationsBook/Labs/coverage/xsuds.pdf

Wang, W., Chen, M., & Tang, M. (1999). *Software architecture analysis--A case study*. Proceeding of the 23rd Computer Software and Application Conference, Washington, DC.

Wang, W.-L., Wu, Y., & Chen, M.-H. (1999, December). *An architecture-based software reliability model*. Proceedings of Pacific Rim Dependability Symposium, Hong Kong.

Yacoub, S., Cukic, B., & Ammar, H. (1999). *Scenario-based analysis of component-based software.* Proceedings of Tenth International Symposium on Software Reliability Engineering, Boca Raton, FL.