Prediction of Software Reliability Using Unified Modeling Language

By

Abdelghani younis abdelghani

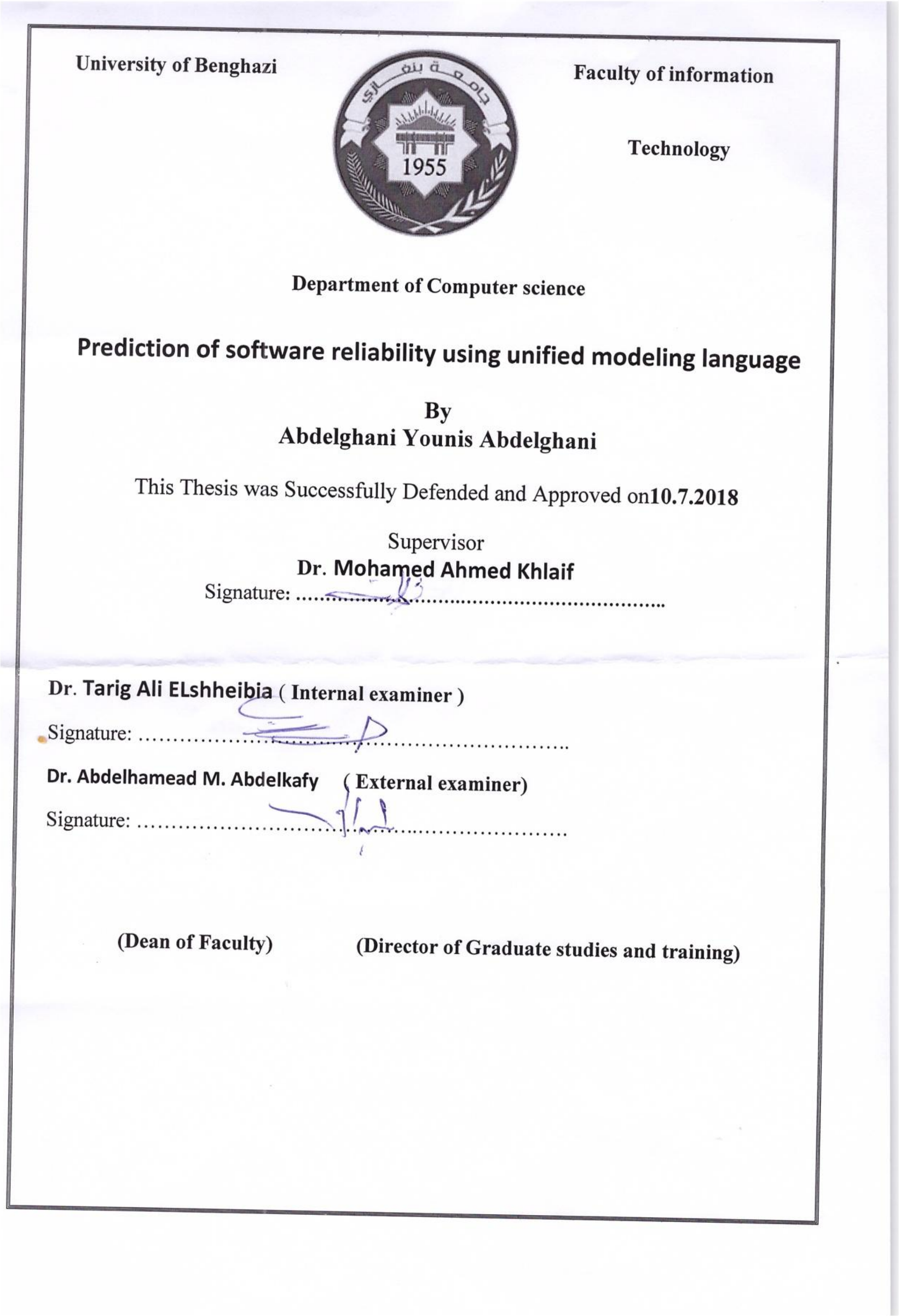
Supervisor

Dr. mohamed khlaif

This dissertation was submitted in Partial Fulfillment of the Requirements for Master's Degree of computer science.

University of Benghazi
Faculty of Information Technology

July 2018



University of Benghazi

Faculty of information Technology

Department of Computer science

Prediction of software reliability using unified modeling language

By

Abdelghani Younis Abdelghani

This Thesis was Successfully Defended and Approved on**10.7.2018**

Supervisor

Dr. Mohamed Ahmed Khlaif

Signature: ....

Dr. Tarig Ali ELshheibia (**Internal examiner**)

Signature: ....

Dr. Abdelhamead M. Abdelkafy (**External examiner)**

Signature: ....

(Dean of Faculty) **(Director of Graduate studies and training)**

DEDICATION

All praise to Allah, today we fold the days' tiredness and the errand summing up between the cover of this humble work.

To the utmost knowledge lighthouse, to our greatest and most honoured prophet Mohamed - May peace and grace from Allah be upon him.

To the Spring that never stops giving, to my mother who weaves my happiness with strings from her merciful heart .

To whom he strives to bless comfort and welfare and never stints what he owns to push me in the success way who taught me to promote life stairs wisely and patiently, to my dearest father.

To whose love flows in my veins, and my heart always remembers them, to my brothers.

To my dear wife
With all the love and appreciation to the companion of my heart that came with me towards the dream step by step we sowed together and harvested together and we will stay together, Allah willing.

To those who taught us letters of gold and words of jewel of the utmost and sweetest sentences in the whole knowledge. Who reworded to us their knowledge simply and from their thoughts made a lighthouse guides us through the knowledge and success path, To our honoured teachers and professors.

Acknowledgement

I would like to thank and appreciate Allah Almighty who guided me to prepare this research.

I wish to express my deep sense of gratitude to my supervisor Dr. Mohamed khlaif, for his outstanding guidance and support which helped me in completing my thesis work.

I would also like to thank Dr. kanz , for her valuable assistance and help to fulfill my work.

Besides my advisors, it is a matter of great privilege for me to present this project to my dissertation external examiner, for corporation and being a part of this work.

Words are inadequate in offering my thanks to miss. hiba, for continuous support and cooperation

Last, but not least, I would like to express my heartfelt thanks to my mother, my brothers for unconditional support and encouragement to pursue my interests, for listening to my complaints and frustrations, and for believing in me, my friends and colleagues for their help and wishes for the successful completion of this project.

LIST OF CONTENTS

Contents **Page No.**

LIST OF TABLES

LIST OF FIGURES

LIST OF Abbreviation

abbreviation	Meaning
IEEE	Institute of Electrical and Electronics Engineers
UML	Unified Modeling Language
SRT-PRO	Software Reliability Professional Tool
OOP	Object Oriented Programming
NHPP	Non-Homogeneous Poisson Process
TBF	Time Between Failure
FC	Fault Count
FS	Fault Seeding
FI	Failure intensity
IDB	Input Domain Based
SRE	Software Reliability Engineering
OP	Operational Profile
SRGMs	Software Reliability Growth Models
DFR	Decreasing Failure Rate
DTMC	Dicrete -Time Markov Chain
COTS	Commercial Off-The-Shelf
LOC	Line Number Of Code
BP	Busy period
CCN	Cyclomatic Complexity Number

LIST OF APPENDICES

Prediction of Software Reliability Using Unified Modeling Language

By

Abdelghani younis abdelghani

Supervisor

Dr. mohamed khlaif

Abstract

As software reliability studies attracted great deal of attention, the current study addresses one of the important challenges of software reliability analysis. White box reliability analysis approach provides useful information that generates more precise decisions by identifying the unreliable and untrustworthy critical parts. Moreover, tracing potential unreliable parts in early phases are difficult due unavailability of executable code.

Theoretically, white box reliability prediction approach supports prediction in the design phase. Therefore, the proposed approach divides the reliability analysis process into six stages, in which design artifacts such use case diagram and activity, sequence diagram are utilized. The proposed approach also predicts the system reliability by estimating the method level failure intensity through the busy periods and complexity weight values. These weight values calculate the probability of an activity being transferred to a complete state using a Markov chain. Furthermore, it is possible to simulate change in the system reliability when the system operational profile is changed.

Practically, experimental study was conducted to evaluate the proposed approach applicability. The results of the study show that technique can predict software reliability more accurately and simulates profile changes.

CHAPTER 1

1 INTRODUCTION

1.1 Background

The increase use of computer-based systems for any application such as medical, nuclear or any critical aspect in modern society require software should be highly developed in terms of quality, which, in turn, should be continually managed and improved. Where, one failure in the system can cause huge loss .Therefore, many efforts have been devoted to enhancement of software quality and focus on an important aspect which is reliability

Institute of Electrical and Electronics Engineers (IEEE) defines reliability as "The ability of a system or component to perform its required functions under stated conditions for a specified period of time " (Khan and Malik,2017). The reliability of software system is measured by the removal of these errors. Most of the software reliability models are based on time between software failures or the number of failures in execution time period. Where, examination of structure is not taken into account, execution time not be the only factor to estimate the behavior of application failure (Lyu, 1996).

The changes related to software system quality such the changes in architecture are costly when the changes take place in later phase of the software development life cycle. Therefore, early assessment of the reliability is very important, But this is difficult given the inadequacy execution information.

When time-based reliability models used the overestimated values were represented in the increase between failure events. Therefore, latter test cases are less likely to reveal faults this means that only depending on the time between failure does not produce acceptable results (Alrmuny,2014).

There are many analytical models for software reliability estimation e.g, the Goel-Okumoto, Jelinski-Moranda and Musa-Okumoto (M-O) models etc, are based on the time domain. Generally, there are restrictions on the current techniques of analysis of reliability as evaluation process delayed to the system test phase, thus major design decisions have already been taken (Krajcuskova,2007), (Everett,1999).

The methods of software reliability analysis can be categorized in two ways : white and black box reliability analysis. The major difference between the two methods is that the former considers the internal structure of the software estimates the system reliability and can be used in earlier stages of software development, especially at design time to identify critical components. The latter method estimates system reliability from failure history that are collected during operation phase and ignoring software structure called a software reliability growth model and The reason of naming it reliability growth is that the models that are used in black box reliability analysis generally assume that bugs are fixed right after they are identified and there is no case of inserting additional bugs during the debugging phase (Krka et al,2009). Measuring reliability or predicting it in earlier allow developers of software engineering to correct errors and enhancement it.

1.2 Statement of problem

Many research efforts have been made to develop models of software reliability, but no single model can be suitable for everyone. The proposed models are based on different assumptions and techniques. Analytical models have been introduced focusing on the data collected during the testing phase and ignoring the structure of the software. This keeps the information hidden about the internal interaction mechanism among the software components. In addition, they use statistical methods that are difficult to apply if there is insufficient data to test.

The research will focus on predicting the reliability of the software earlier so there will be more flexibility for the developers in making design decisions and determining the parts that need reviewing rather than looking at the system as a whole (Alrmuny,2014).

1.3 Aims of the study

With the growing complexity of applications reliability analysis. Therefore, the research in the area of software reliability analysis has gained prominence. So, The aims of this dissertation as following :

1- The primary objective of this study is to introduce the ability of software reliability early prediction.

2- Explore architectural alternatives based on component reliability.

3- Analyze the sensitivity of the application reliability**.**

4- Relate application reliability to its architecture and individual component reliabilities.

Finally, this dissertation to introduce a technique that will support early applicability. The results of this dissertation can help the practitioners of software reliability prediction in earlier to choose the appropriate methods for quality assurance.

1.4 objectives of the study

Most of the existing models have been criticized for being too detailed or complex. The approaches developed in this work provide basis to solve problems of software reliability prediction and The objectives of this study are given below :

1- To identify bottlenecks reliability for each components.

2- To simplify the process of reliability tracing by the parts of software rather than all.

3- To improve the quality of the software through enhancing reliability of the component.

4- To make analysis easy to trace the changes of reliability according to the operational profile.

This prediction uses design artifacts Unified Modeling Language (UML), which is able to extract in the early phase.

1.5 Methodology

In order to develop highly reliable software in an effective manner, the analysis should be performed in the early stage of the software development life cycle that requires the following:

1- Investigate previous researches and the literature review that related with my work in order to focus on white box reliability prediction aspect.

2- Collect the requirement documents and architectural specifications that related with the case study.

3- Analysis and Design the case study to extract the artifacts based on UML (Bell,2003).

4- Construct the proposed approach to extract the parameters and data to analysis.

5- Comparing the result with previous work.

1.6 Scope and limitation

The study covers white box reliability analysis in terms of inter-component interactions in operational profile by using (UML) diagrams to express component relationships when the software system is developed with object oriented programming (OOP), The study will not cover the failure behavior models based on the test time information that collected during the system testing phases.

1.7 Significance of study

White box reliability analysis uses the software's internal information and early artifacts such as requirements and architectural specification to predict the reliability in the early phase. It is will help developer before principal design decisions are made to improving the quality of software.

1.8 Structure of the dissertation

The study is organized as follows :

Chapter 1: Includes Introduction, which contains the subject background and determines the context of the dissertation in the statement of the problem, aims and objectives of the study as well as limitations and methodology that will be followed;

Chapter 2: Familiarizes the concepts and methods related to the software reliability and the previous studies related to the subject and Give a glance about white and black box reliability analysis;

Chapter 3: Represents the core this study and offers the approach that attempt to overcome the cons of the existing models;

Chapter 4: Includes the focuses on experimental case study; and comparative the results with other.

Chapter 5: Concludes the study by show the obtained results.

CHAPTER 2

2 LITERATURE REVIEW

2.1 Historical Background

The estimation of software reliability used statistical models such as historical data of similar projects or organizations or direct software measures (Blischke and Murthy,2011).

Software reliability correctness has been highlighted as early as 1975- 1976 by Parnas (Parnas,1975). Black box method is Prevalent and Several critiques have appeared in the literature one of this ignore information and reliabilities about of the components (Hamlet,1992). And the examples of software reliability models are Jelinski-Moranda Model, Generalized Goel NHPP Model (Tausworthe and Lyu,1996), and Goel-Okumoto Model, Verrall Model (Yang and Chao,1995). These models have advantages and disadvantages and specific assumptions.

Dimov et al. (Dimov et al,2010) use testing methods to generate data for reliability analysis from small survey, but without much detail of test results to reliability model parameters. Moreover, it focuses on testing of existing systems. Chen et al (Chen et al,1992). Due to the saturation effect add structural coverage to traditional time-based software reliability until excludes test cases that do not increase coverage.

Murphy et al. (Murphy and Gent,1995). Focuses techniques of systems already deployed such as questionnaires, customer service calls or bug reports and does not discuss derive reliability model parameters. Mannhart et al. (Mannhart et al,2007), compare available methods for modeling expert judgment, and discuss their limits when applied to software reliability prediction.

Goševa-Popstojanova and Trivedi’s white box reliability models divide into path-based, state-based, and additive models. Paths of execution is original source of analysis of first, calculating the possibility a component transferred to other component, additive models do not consider software architecture explicitly (Goseva,2000).

The problem, in the literature consigning the estimation used at later stages of software development. Therefore, Software reliability prediction techniques are important at early stages of development life cycle, over the same data.

We are interested in studying the reliability because we believe Unreliability has a number of consequences, as poor reliability can have negative implications on Safety, Cost of repair, maintenance and Reputation.

2.2 Software reliability analysis

In this respect, researcher will review aspects related to research in the field of software reliability, in terms of reliability analysis, which includes a look at the causes of program and data failure as well as the relationship, measuring and modeling the reliability of the software. Additionally, he will classify the models and reliability parameters, which deemed one of the most important pillars of measure the reliability, along with the operational profile and its relationship with reliability. Finally, he will review the reliability analysis in terms of both black and white reliability analysis boxes.

2.2.1 Salient features of software failures

- Each application at least unique and a little differences in the code may mean large differences in the behavior of the application.
- Application faults are caused by hidden design flaws. So, application faults are static and exist from the day the application was written until the day they get fixed.
- Application reliability depends on the amount and quality of corrections not on time.
- Commercial software application of 350000 lines of code can contain over 2000 programming errors. In other word, average of six software faults for every 1000 lines of code written, that is Result of a research study.
- The significance of the fault affects repair time: a more significant fault is prioritized and corrected promptly, whereas an inconsequential bug may be left to stay in the system for the whole of its life cycle.
- Software in huge systems is inversely proportional to application size.
- When the application is deployed in the operational phase it is usually installed in many places and operational conditions differ from place to place.

Therefore, failure data, if collected, comes from different sources (Karanta,2006).

2.2.2 Failure data

When we are talking about failure data, we define these terms and other related software reliability terminology. To prevent confusion in the rest of our work, we will adhere to the definitions.

- Failure occurs when the user perceives that a software program ceases to deliver the service or occurs when the delivered service deviates from the correct one.
- Fault is uncovered when either a failure of the program occurs, or an internal error is detected within the program. The cause of the failure or the internal error is said to be a fault. It is also referred as a “bug”.
- Error service failure means deviation of an external system state from the correct system state. This deviation is called an error. **F**igure 2 indicate the mutual relationship between them (Avizienis et al,2004).

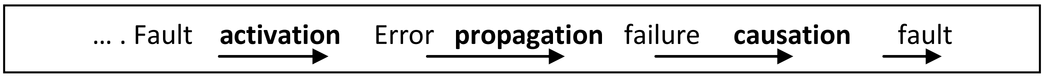


Figure 2 : Relationships between failures and errors (Avizienis et al,2004).

2.2.3 Software Reliability Measurement and Modeling

Software reliability measurement includes estimation and prediction

- Estimation used statistical inference techniques to failure data that obtained during system test, This is a measure regards the achieved reliability from the past until the current point. This technique is suitable for testing the system or an operational stage. In other word, When failure data are available the estimation techniques can be used.
- Prediction is an activity determines future software reliability based on available software metrics and it is used **w**hen failure data are not available and prediction involves different techniques (Shanmugam and Florence,2012).

2.2.4 Classification of Models

Software reliability models are divided based on failure history and data requirements, respectively (Shanmugam and Florence,2012).

- Failure History: This type can be classified according to the nature of the failure process studied as indicated below.

- Time between failure models (TBF Models): The process under study is the time between failures. It is assumed that the time between (i-1) th and (i)th failure is a random variable. There are some failure rate models such as : Jelinski and Moranda Model, Schick and Wolverton Model and Goel and Okumoto Imperfect Debugging Model.

- Fault count models (FC Models): The random variable of interest is the number of faults (failures) occurring during specified time intervals. And The key models in this class are Shooman exponential model, Musa execution time model and Discrete reliability growth model.

- Fault seeding models (FS Models): A Program has unknown number of indigenous Faults. To this, a known number of faults are seeded.

- Input domain based models (IDB Models): In this approach, a set of test cases is generated from the input covering the operational profile of the input. Usually the input domain is partitioned into a set of equivalent classes, each of which is usually associated with a program path.

- Data requirements: They can be grouped into two main groups as Empirical Models and Analytic Models.

- Empirical Models: An Empirical model develops relationship or a set of relationship between measures and a suitable software metrics such as program complexity using empirical results available from past data.

- Analytic Models: They requires some form of data gathered from software failures. It is based on fitting of a suitable distribution with required assumptions for simplicity on a set of data gathered during software testing. Figure 2.1 show classification of software reliability models**.**

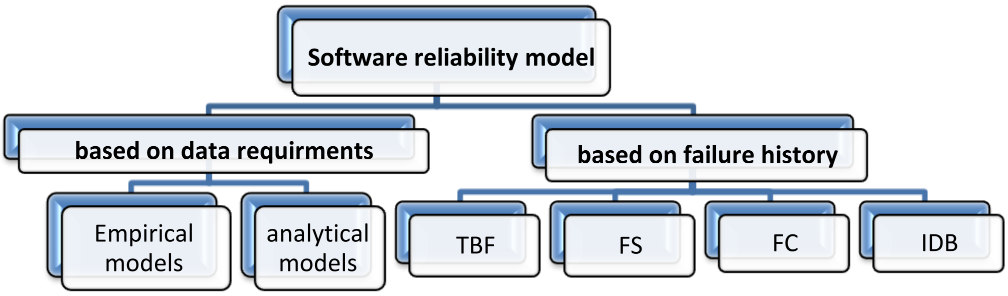


Figure 2.1 : shows the classification of software reliability models (Shanmugam and Florence,2012).

2.2.5 Reliability parameters

The input parameters of architecture-based reliability analysis are divided into three categories: failure parameters, behavioral parameters and execution environment parameters.

- Failure parameters: they describe the failure behavioral of an element (system, components, scenarios, methods etc). There are three types of failure models (Gokhale and Trivedi,2006), (Gokhale,2007).

- Probability of failure (reliability) R: It is the most frequently used parameter and it is the probability that software will cause a failure of a system. So, essentially treats the components (and other elements) as black boxes.

- Constant failure rate λ : It is more accurate than probability of failure and defined as the number of failure occurrences per unit of time. Therefore, it can consider time spent in the component during the execution.

- Time-dependent failure intensity $\lambda(t)$: It is account for the dependent executions of components in case of loops and defined as a rate of change of expected number failures with respect to time.

- Behavioral parameters: The behavioral parameters model, the operational profile of the system and it specification is a challenging process, especially at design time. The information for the specification can be gathered by profiling, by collecting the software usage statistics or partially by studying behavioral unified modeling language diagrams (Brosch et al,2012).

- Execution environment parameters: Some reliability prediction approaches consider the execution environment, in which the system is deployed. And the execution environment parameters are often supplied by hardware vendors and infrastructure providers (Distefano and Puliafito,2009).

2.2.6 Operational profiles

Software can fail due to the inputs it receives from the external environment. So, The reliability of a software-based product depends on how the computer and other external elements will use it. The reliability estimate depends on testing the product as if it were in the field. The operational profile (OP), a quantitative characterization of how the software will be used, is therefore essential in any Software Reliability Engineering (SRE) application. Developing an operational profile for a system involves one or more of the following five steps (Musa,1993) :

- Customer profile: Customer profile consists of an array of independent customer types and is the individual, group or organization, each of these types of customers may be expected to utilize the spreadsheet in a substantially different way. The customer profile is the list of customer types and the associated probabilities. These probabilities are simply the proportions of time that each type of customer would be using the system.
- User profile: Users of systems may be different from the customers of application product. A user is a person, group, or institution that operates, as opposed to acquire, the system. The user profile is the set of user types and their associated probabilities of using the system.
- System-mode profile: System mode is a way that a system can operate. Most systems have more than one mode of operation. For example, system testing may take place in batch mode or user-interactive mode.
- Functional profile: After a good system mode profile has been developed, the focus should turn to evaluation of each system mode for the functions performed during that mode, and then assigning probabilities to each of the functions. Functions are essentially tasks that an external entity such as a user can perform with the system.

- Operational profile itself: Determine the elements involved in determining operational profiles from functions. A function may comprise several operations. In turn, operations are made up of many run types. Grouping run types into operations partitions the input space into domains. A domain can be partitioned into sub domains, or run categories. The process of operational profile stages can be shown in Figure 2.2

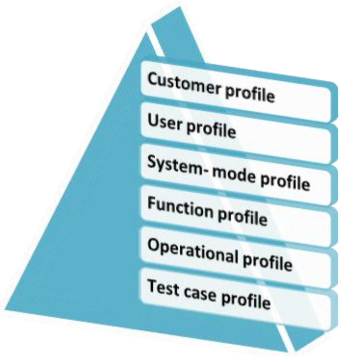


Figure 2.2 : The stages of the operational profile (Musa,1993).

2.2.7 Black-box reliability analysis

They focus mostly on quantification of failures and down-times and employed in later stages of software development or they are used on systems that are already deployed. This type of models analyze the reliability of the whole application while ignoring its internal structure. The main representative of this type is Software Reliability Growth Models (SRGMs) (Aggarwal and Gupta,2014). And Figure 2.3 shown Defect detection rates with time

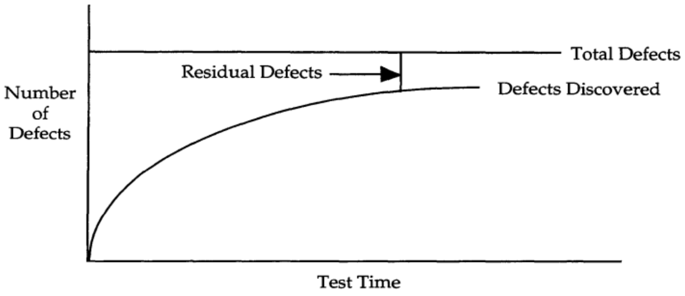


Figure 2.3 : Residual Defects (Aggarwal and Gupta,2014) .

- Software reliability growth model: A software reliability growth model is one of the fundamental techniques used to assess software reliability quantitatively. Software reliability growth models can be used as an indication of the number of failures that may be encountered after the software has shipped and thus as an indication of whether the software is ready to ship. Thus, SRGM is used to determine when to stop testing to attain a given reliability level, and These models attempt to statistically correlate defect detection data with known functions such as an exponential function. Therefore, have a parameter that relates to the total number of defects contained in a set of code. If we know this parameter and the current number of defects discovered, we know how many defects remain in the code. Figure 2.3 , shows the number of residual defects that helps us decide whether the code is ready to ship and how much more testing is required if we decide the code is not ready to ship(Kashyap and Rana,2015), (Mohd and Nazir,2012).

Research efforts in software reliability engineering have been conducted over the past three decades. As result, There are many software reliability growth models, and many ways to represent the data that is used to create those models, and some researchers believe that each organization needs to try several approaches to determine what works best for them.

Software reliability growth models have been grouped into two classes of models concave and S-shaped. Both the s-shape and concave curve depict the asymptotic behavior i.e. a finite asymptotic value is attained by both the curves because the fault rate plunges down steadily as the defects are detected and repaired during the tenure of testing.

- Concave shaped models: The Concave shaped models are Decreasing Failure Rate (DFR) models. In these models the failure rate decreases at a constant pace as the number of faults are detected and removed. The idea behind DFR is that as the given predetermined number of errors are detected and removed, the software reliability improves. In these models when failure data is supplied as input, the failure rate reduces steadily and becomes constant after some time, during the testing tenure. The constant decrease in the failure rate is attributed to regular detection and removal of the faults at a constant pace during the testing. Goel-Okumoto, Musa and Jelinksi-Moranda models are amongst the earliest concave shape models.
- S- shaped models: The models depicting S-shape patterns also demonstrate the asymptotic behavior similar to the concave model. The failure data which

is used to plot the curve is analyzed in two phases of software testing. In the early phase, the testing is comparatively less effective than the later phase because the testing team performs testing using the same test cases as used by the development team, therefore the failure rate decreases. This is the reason why the curve attains the inward bulge. Later on, in the application testing phase, the new defects are uncovered. Yamada Weibull Effect, Pham and Nordmann models are amongst the earliest S- shape models. Figure 2.4 shows the difference between the two models (Mohd and Nazir,2012).

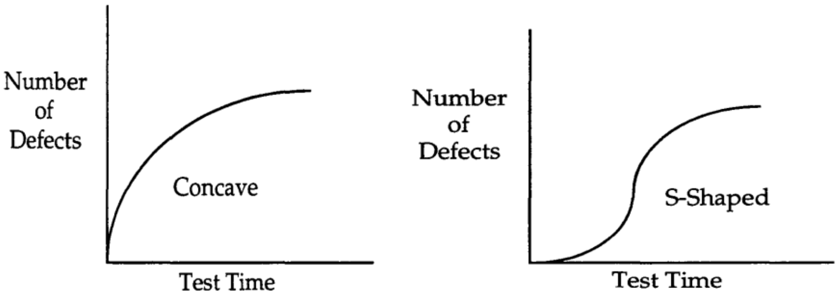


Figure 2.4 : Concave and S-Shaped Models (Mohd and Nazir,2012).

- Test code coverage: Is a measure that describes the degree to which certain elements of the source code have been tested. In other word, it has been used as an indicator of testing effectiveness. It proposed as a possible solution for some drawbacks of SRGMs. The larger part of software's structure is exercised by tests, the more faults will be detected and reliability will grow (An and Zhu,2010).

The technique can be applied as source code instrumentation (compiled and the test cases are executed) and data collection can be done automatically by specialized code coverage tools. The code coverage is measured by four code coverage criteria are block cover, decision coverage, C-use and P-use. And, coverage per test case, according to equation 2 (Gokhale and Trivedi,1999) :

$$\text{X coverage} = \frac{\textbf{Number of x covered by test case}}{\textbf{Total number of x}} \cdots\cdots\cdots\cdots\cdots\cdots (2)$$

Where, x is the given coverage criterion and The basic coverage criteria are Statement (or block) coverage, Branch (or decision) coverage, C-use coverage and P-use coverage.

(SRGMs) have been used to estimate reliability by using the time dependent failure data. When, these models were used notable overestimation of reliability was observed. Thus, the fact that latter test cases are less likely to reveal faults that reside in uncovered portions of the code, a saturation state occurs (Alrmuny,2014).

2.2.8 White-box reliability analysis

In order to predict the system reliability in the early phase, the available sources, such as requirements documentation and design diagrams are processed to extract a failure model for the system. To determine the characteristics of erroneous behaviors. The system failure model is then combined with the architectural specification and reliability parameters estimation to finally produce system reliability estimation (Krka et al,2009). Reliability prediction process can be shown in Figure 2.5.

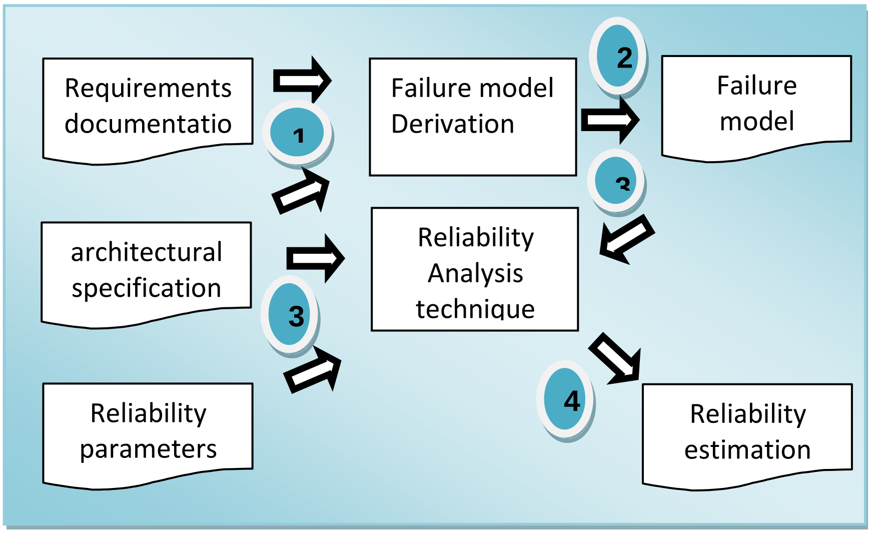


Figure 2.5 : Scheme of the architecture based analysis process (Krka et al,2009).

Based on the way the architectural model is mapped to a formal model, white box reliability estimation models can be classified into three major types : path based, state based and additive based models (Goseva,2000). The classification of architecture-based software reliability models by Figure 2.6 (Gokhale,2007).

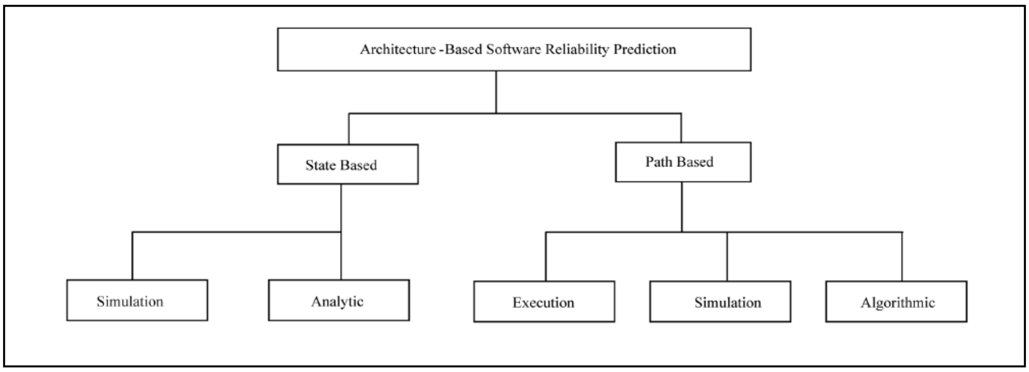


Figure 2.6 : classification of architecture-based software reliability.

- Path-based models: A path is an independent sequence of components or statements to carry out a system function (Yacoub et al,1999). The architecture of the application is represented by enumeration of the possible execution paths through the application, Path based models easy to get information if the software is already implemented. In spite of that, it is not easy to analyze all execution paths before implementation (Cortellessa and Cukic,2002), (Rodrigues et al,2005).

The count of paths can be done, by simulation or by analysis of scenarios based on UML sequence diagrams. Where the nodes represent the components and edges represent possible transitions between the components. Figure 2.7 shows a system function with a path (path N) that executes the {1,3,4,7} components in order.

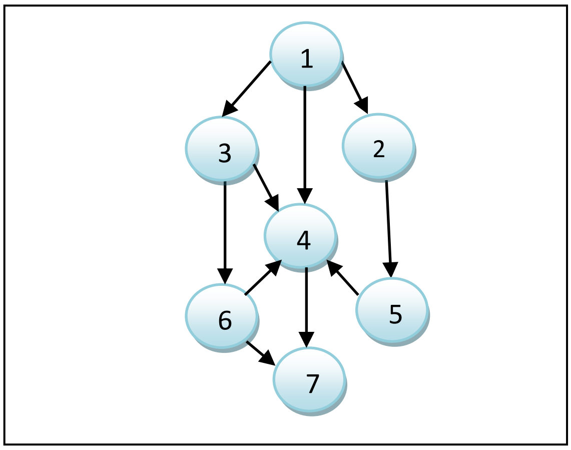


Figure 2.7 : path based model .

shooman model is one of the representative path based models (Shooman,1976). It is assumed that the probability of failure for a path (f) and the frequency of the execution path (q) are known. The accumulative failure number in N system executions is calculated, according equation(2.1) as follows:

$$nf = Nf1q1 + Nf2q2 + \cdots + Nfkqk = N\sum_{i=1}^{k} fiqi \quad \ldots\ldots \ (2.1)$$

Number of paths N close infinity. Thus, the probability of failure of an execution run is given according equation(2.2)

$$Qs = \lim_{n\to\infty}\frac{nf}{N} = \sum_{i=1}^{k} fiqi \quad \ldots\ldots\ldots\ldots \ (2.2)$$

the reliability of system Rs according equation(2.3) as follows:

$$Rs = 1- Qs \quad \ldots\ldots.. \ (2.3)$$

The biggest problem of path-based approaches is not easy for analyzers to predict all execution paths before doing implementation. Another problem occurs when there is a loop on the execution path may lead to infinite paths.

- State based models: The estimate system reliability in state based models by showing individual components as individual states, and calculating the possibility of one component being transferred to other component. The transition probability between components through operational profile (Gokhale and Trivedi,2002),(Reussner et al,2003).

State based models include the failure state (F) and the complete state (c), edge that transfers to a complete state (c), and edges that are transferred as failures from all the components, are added. The transfer possibility of edges that lead to the failed state is assigned as 1-R_i.

The underlying state space model can have several formal representations. The most frequently used is a Dicrete-Time Markov Chain (DTMC) to find the possibility of the system(Goševa and Trivedi,2001).

The problem of State based models, when the number of state increase because of an increased number of components, the number of interactions happening between components increase. Therefore, causing a state explosion, It becomes difficult to analyze a large software system, and The example of state-based model shows in Figure 2.8.

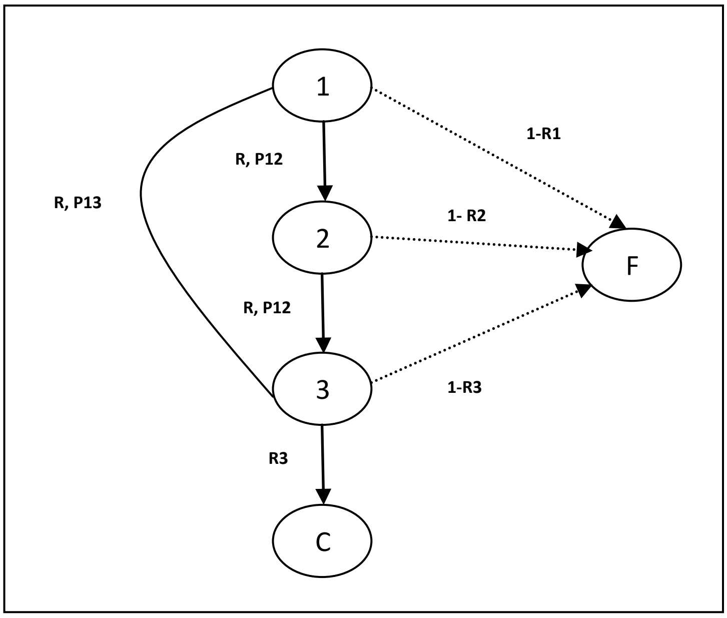


Figure 2.8 : state based model .

- Additive based models: This type consider the software architecture only implicitly, do not consider software architecture explicitly (Goseva,2000), (Everett,1999). Divide the system into subsystems, and each sub-system is measured separately, where it is assumed that all the sub systems are tested thoroughly, by adding all the sub system failure rates λs(t) as follows:

$$\lambda s(\mathrm{t}) = \lambda 1(\mathrm{t}) + \lambda 2(\mathrm{t}) + \ldots\ldots + \lambda \mathrm{n}(\mathrm{t}) \ldots\ldots\ldots\ldots\ldots \quad (2.4)$$

The problem of this model assumes that all the sub system are operated without architecture information exchanges.

The most of the existing models are generally useful, but they have limits. Do not consider analyzing of reliability in the early phase, because a lack of execution information and difficult to predict all execution paths without operating the software. Difficulty in get quantitative results. Therefore, make the software quality better will be limited.

CHAPTER 3

3.1 Observation

The state based model is suitable when analyzing systems, it expresses their component calling relationship with a call graph, but when the components expand, this method does not apply the Object Oriented Programming (OOP) features.

That component interaction works as a sequential process in the existing model. In other hand, normally several components interact with each other to carry out a certain task in the OOP. Figure 3 indicate the work in a bidirectional manner among components.

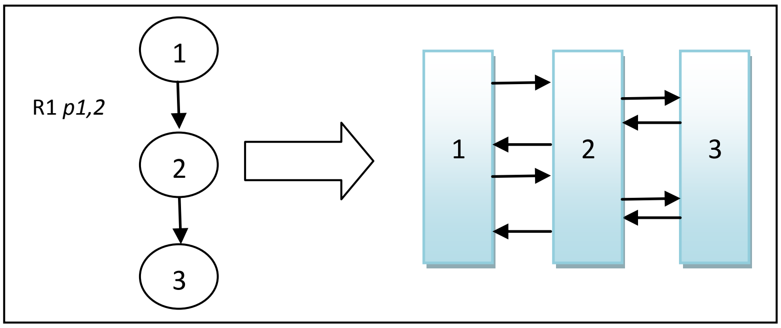


Figure 3 : The interaction of component.

The component transition always has the same context in the existing models. But the process of actual components varies depending on the kind of public interface the component provides. Thus, method level, must be taken into account. Figure 3.1 shows different usage of components.

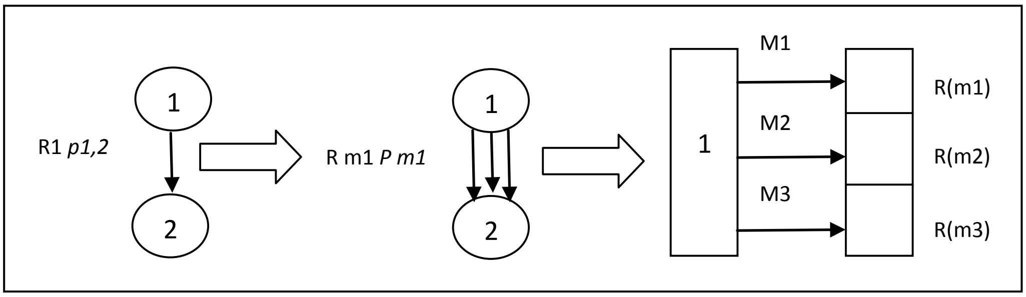


Figure 3.1 : Component different usage.

3.2 Approach Overview

Considering the factors mentioned earlier, The proposed approach consist of six stages where, first three stages focus on analysis the system to assigning the usage probability for each level by using UML diagram such as use case diagram, the activity diagram and sequence diagram. So, many of artifacts in the early phases of the life cycle provided by UML diagrams, and the rest of stages focus on calculating the component failure rates. Finally, we calculated the reliability and probability to estimate reliability of each level.

3.3 Stages of the approach

3.3.1 Identify use case and actor

In stage1. We utilize use case diagram to show the operational profile. The operational profile has been defined in chapter2. Use case diagram is a graphic depiction of the interactions among the elements of a system. It is possible to distinguish actor and use case. The former refers to the systems user and the later refers to the system usage, then indicate their association (Bell,2003). The indication **[Pactor]** is possibility that each actor will utilize the system, **[Passociation]** is the possibility that each actor will utilize the use case, Through these, we infer the likelihood of execution of one use case scenario which is **[Puc]**. Figure 3.2 shows the operational profile in use case diagram. Equations 3, 3.1, 3.2 show the rules of calculation

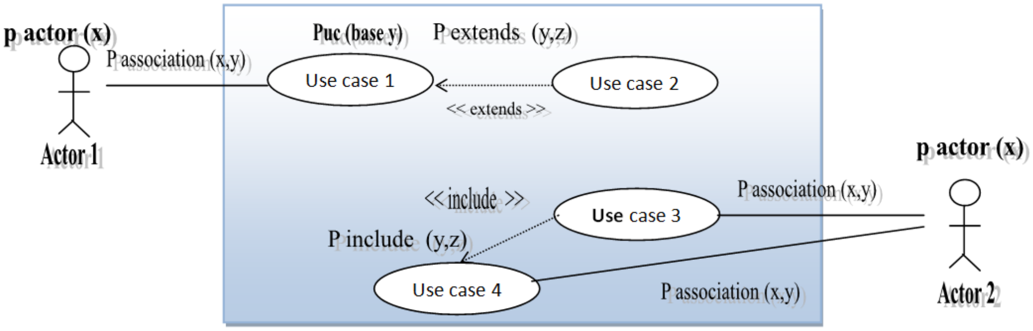


Figure 3.2 : System in Use Case Diagram.

Where :

$$\sum_{x=1} Puc(x) = 1 \qquad \cdots\cdots\cdots\cdots\cdots\cdots(3)$$

$$\sum_{x=1} Pactor\ (x) = 1 \qquad \cdots\cdots\cdots\cdots\cdots(3.1)$$

$$\sum_{x=1} Passociation\ (x, y) = 1 \qquad \cdots\cdots(3.2)$$

In addition, in UML modeling there are more types of relationships such as inclusive relationships and extended relationships. The relationships can be added to the model when use case is in common in to two or more use case. According to equation 3.3.

$$\mathbf{P}include\ (x, y) = 1 \qquad \cdots\cdots\cdots\cdots\cdots\cdots(3.3).$$

Moreover, the developer can use the relationship to identify that one use case extends the behavior of another use case. Extension is a directed relationship that specifies how and when the behavior defined in usually supplementary (optional). Extending use case can be inserted into the behavior defined in the extended use case. Extract relationships can be added to a model.

The probabilities of each type of use case are according to the equations 3.4, 3.5 and 3.6 are as follows:

$$\mathbf{P}\ \mathrm{uc(base\ y)} = \sum\nolimits_{\mathbf{x=1}} \{\mathbf{P}\ \mathrm{actor\ (x)} * \mathbf{P}\ \mathrm{association\ (x, y)}\} \qquad .... \ (3.4)$$

$$\mathbf{P}\ \mathrm{uc(include\ z)} = \sum\nolimits_{\mathbf{x,y=1}} \{\mathbf{P}\ \mathrm{actor\ (x)} * \mathbf{P}\ \mathrm{association\ (x, y)} * \mathbf{P}\ \mathrm{include\ (y, z)}\}.(3.5)$$

$$\mathbf{P}\ \mathrm{uc(extend\ z)} = \sum\nolimits_{\mathbf{x,y=1}} \{\mathbf{P}\ \mathrm{actor\ (x)} * \mathbf{P}\ \mathrm{association\ (x, y)} * \mathbf{P}\ \mathrm{extends\ (y, z)}\}.(3.6)$$

The calculation of use case probability take all these relations into account according to the equation 3.7.

$$\mathbf{P}\ \mathrm{use\ case(x)} = \mathbf{P}\ \mathrm{uc(base\ x)} + \mathbf{P}\ \mathrm{uc(include\ x)} + \mathrm{P\ uc(extend\ x)} \quad .... \ (3.7)$$

The proportion of derived probability in the system, Pnorm(x), is the probability that will be used by each use case, the value can be obtained by means of equation 3.8.

$$\mathbf{p}\ \mathrm{norm\ (x)} = \frac{\mathbf{p}\ \mathrm{usecase\ \ (x)}}{\sum\ \mathbf{p}\ \mathrm{usecase\ \ (all)}} \qquad .... \ (3.8)$$

3.3.2 Identify probability of activity

In stage2. We analysis each use case to show its activity, with assumption that one use case scenario has one key activity list. Activity list is the procedures of processing applied use case scenario, This activity is process unit in the test. The probability of the activity being spread to the next activity. The model derives the probability of activity transition possibility that includes multiple components. Figure 3.3 indicate identify activities.

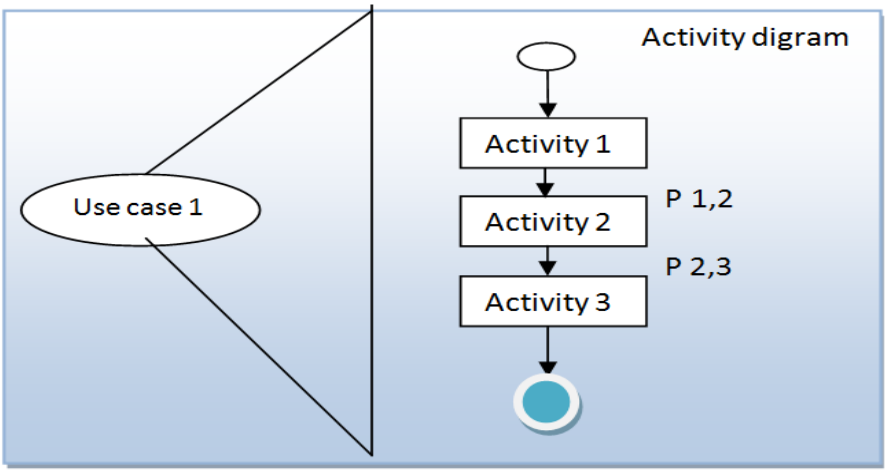


Figure 3.3 : Identifying Activities

3.3.3 Identify component Interaction

Stage3. In this stage we can determine interaction within and between components by using a sequence diagram, each component that can be estimated was utilized by counting the time methods of component which named (busy period). Figure 3.4 shows the process of interaction between the components and the busy period.

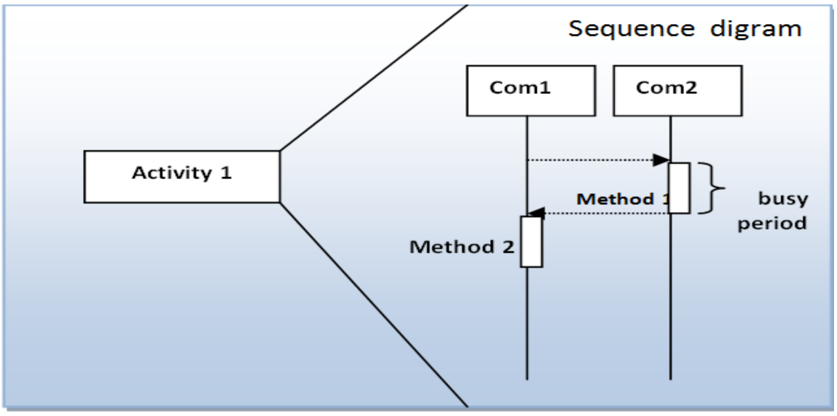


Figure 3.4 : Component Interactions .

3.3.4 Calculation of Component Reliability.

Stage4. Derive the sequence diagram reliability with utilized component method level failure rate, and call count of each method (BP). The component failure rate May be known through, either historical data, additive model or commercial off-the-shelf software (COTS), We need to assign the method level failure rate with the component failure rate, as the proposed model requires that. In the additive model component failure rate λt (t) at time t can be assigned as in equation 3.9.

$$\boldsymbol{\lambda}(\mathrm{t}) = \boldsymbol{\lambda}1(\mathrm{t}) + \boldsymbol{\lambda}2(\mathrm{t}) + \ldots\ldots..+\boldsymbol{\lambda}\mathrm{n}(\mathrm{t}) \text{ .... } (3.9)$$

The method failure rate **(θ)** can be assigned by multiplying the component failure rates by complexity weight value. Consequently, statement line number **(**LOC**)** per method, or cyclomatic complexity can be used for the complexity weighted value. Table 3 shows the method failure rate derivation.

Table 3 : Expected method failure rate .

Component	**Component Failure Rates(f)**	**Method Name**	**Complexity Weight Value(w)**	**Method Failure Rate(θ)**
Class A	……..	Method 1	…….	W* f
		Method 2	…….	W* f

We can derive the method failure rate for the scenario, according to equation 3.10 as follows:

$$\mathrm{Mf_I} = 1\text{- } (\mathbf{1} - \boldsymbol{\theta}\mathbf{I})^{\boldsymbol{bp}} \text{ .... } (3.10)$$

Finally, method failure rate for a scenario and component level reliability for all components are derived from the equations 3.11, 3.12, respectively. Table 3.1 shows the calculation process (Cortellessa et al,2002).

Table 3.1 : Expected method failure rate in a scenario.

Method Name	**Method Failure Rates(θ)**	**Busy Period Count**	**Method Failure Rate in the scenario MfI = 1- $(\mathbf{1} - \boldsymbol{\theta}\mathbf{I})^{\mathbf{bp}}$**
Method 1	…..	…..	……

Component Failure rate in the Scenario $\boldsymbol{cfi = \sum mfi}$ ……….. (3.11)

Component Level Reliability $\boldsymbol{Rscenario = 1 - \sum cfi}$ ……….. (3.12)

3.3.5 Failure rate prediction.

The probability for each activity transfer to the next activity happening after the scenario reliability (stage 4) (Singh et al,2001). To get the probability of the activities in the key activity diagram being executed and finished correctly, we add complete and failure state. And derive the probability 1-Ri, which is the probability of transfer happening from each activity (i) to failure. Figure 3.5 shows DTMC activity diagram.

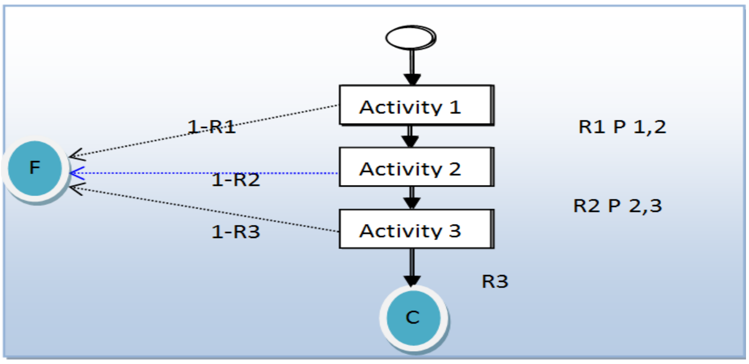


Figure 3.5 : Markov Chain for Activity Diagram

The probabilities of being transferred to the failure state and the component state can be calculated with DTMC, as shown in table 3.2.

In addition, **Rusecase** is the probability of all activities in the key diagram being successfully operated and transferred to complete becomes the probability of one use case being successfully operated.

Table 3.2 : Transition Probability Metrics (Kashyap and Rana,2015) .

	C	**F**	**N1**	**N2**	**…………..**	**Nn**
C	1	0	0	0	…………..	0
F	0	1	0	0	…………..	0
N1	0	1-R1	0	R1,P12	…………..	R1,P1n
N2	0	1-R2	0	0	…………..	R2,P2n
….	…	….	……	…..	……………	……..
Nn-1	0	1-Rn-1	0	0	Rn-1,P(n-1)j	Rn-I,P(n-1)n
Nn	Rn	1-Rn	0	0	0	0

3.3.6 Reliability prediction of System

Finally, the reliability of the system can be achieved through multiplying Pnorm by Rusecase. Figure 3.6 show the process and equation 3.13 as follows:

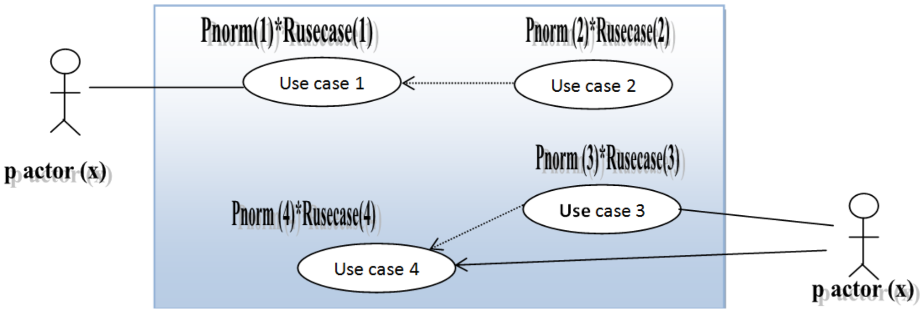


Figure 3.6 . System Reliability Calculation

$$\boldsymbol{R}system = \sum\{\boldsymbol{P}norm(x) * \boldsymbol{R}usecase(x)\} \text{ .... } (3.13)$$

Where, Pnorm is a normalization of each use case’s execution probability and Rusecase is the use case successfully operating.

CHAPTER 4

4.1 The goal of experiment

Here, we carried out a case study of software system to find out whether the suggested approach was valid, and the technique is applicable in the early stage or not. Then the predicted result will be compared to the reliability derived by the black box model in the actual testing phase.

4.2 The environment of experiment

Parking garage automation (reserve your spot) (Edwards et al). Figure 4 shows the software in general. The system will allow customer to place online reservations that include date, time and duration of stay. The garage is also being remodeled so that the parking decks above ground level will be accessible only by an elevator that will lift vehicles to different decks. The garage relies on camera based license plate recognition software to track vehicles as they enter and exit the garage. Additionally, the garage also employs sensors on the parking spots to recognize which spots have been occupied and which is free.

If the software cannot recall the necessary information or if the license plate recognition software is not able to read the license plate, the elevator will not function and the software would prompt the customer to manually input their membership number at the terminal next to the vehicle elevator for it to proceed.

If a registered customer forgets to make a reservation and decides to use the garage, he may be allowed to take a walk-in parking spot without a registration if there are any available spots. These types of customers are known as walk-in customers. If the software recognizes the vehicle registration number but cannot find an existing reservation to the customer who owns the vehicle the customer will have to specify the expected duration and time of departure using the terminal at the vehicle elevator.

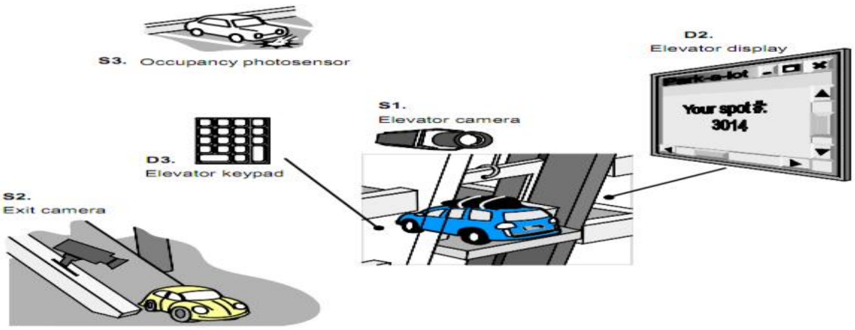


Figure 4 : system application.

In order to restrict people from making reservations they cannot meet, the system has broken down reservations into two groups, confirmed and guaranteed. A confirmed reservation is when a registered used places a reservation, but does not have a credit card on file. A guaranteed reservation is when a registered customer has done the same, but has a credit card on file and uses it when placing their reservation.

If a customer with a confirmed reservation fails to show up after reserving a spot, the spot will be held reserved for a 30-minute grace period, in during which the customer can park on his reserved spot and be billed for the full reserved period. If the customer does not show up to claim his spot during the grace period, the parking spot will be marked unreserved. With a guaranteed reservation, the customer can arrive to their spot anytime during the requested interval and will be charged to their card for that interval. Figure 4.1 show parking garage automation use case diagram. And Figure 4.2 show system sequence diagram for use case 1.

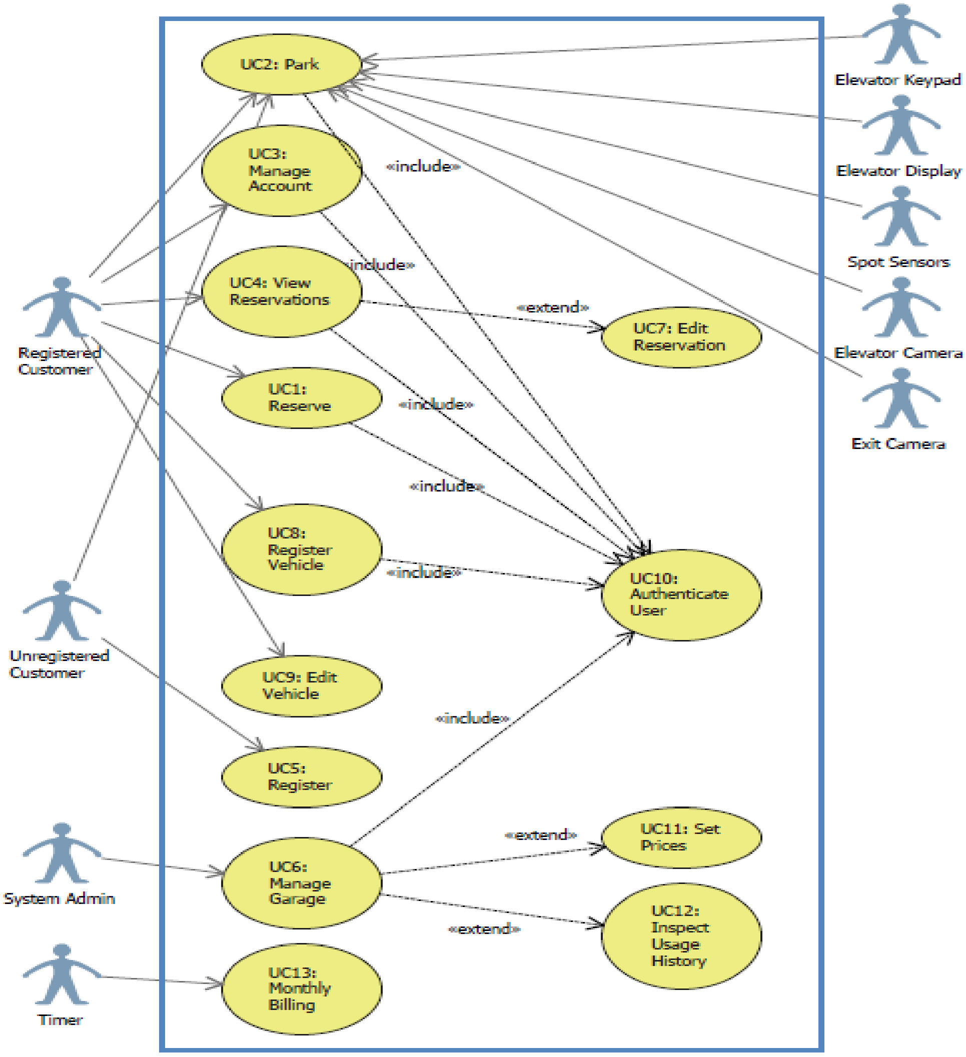


Figure 4.1: Parking Garage Automation Use Case Diagram

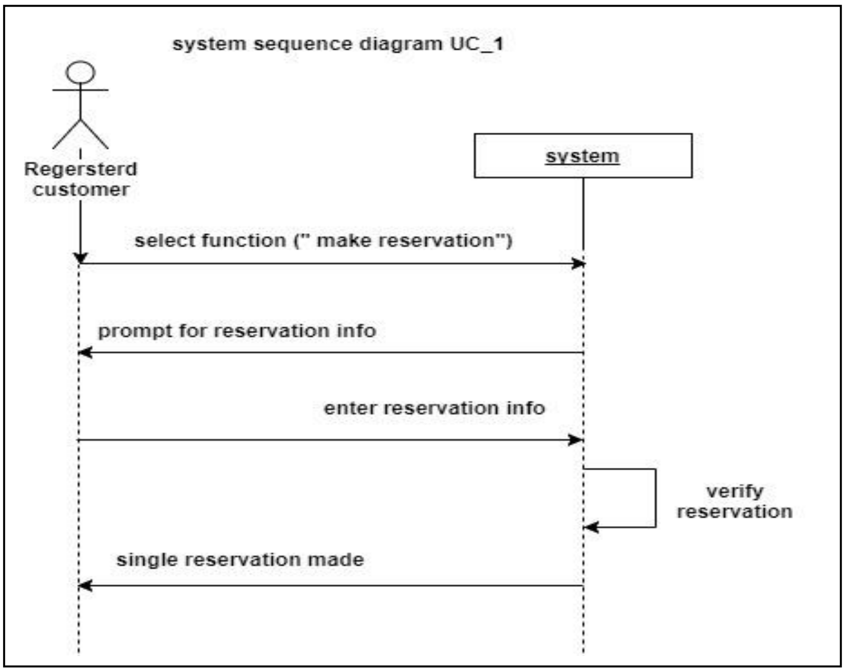

system sequence diagram UC_1
Regersterd customer
system
select function (" make reservation")
prompt for reservation info
enter reservation info
verify reservation
single reservation made

Figure 4.2: system sequence diagram for use case 1.

The main objective is designing a sophisticated system that maximizes occupancy and profit while allowing the customer to easily get access to his vehicle. This means the equipment design and even components of the product should follow a fully described documentation process and the device should meet strict standards of documentation, developmental testing, production testing, and maintenance. If there is wrong or system halt due to software calculation error or bugs, the system can automatically shutdown. As a result, the system guarantees high reliability.

4.3 Stages of the experiment

The experiment was carried out as shown in Figure 4.3. The first step is to extract the design artifacts, such as the use case, sequence and activity diagram based on requirements, and architectural specification.

The test target have been formed with 100 revision on the source repository and chose the list of modification changes that happened due to bug/fix to count each revisions and each classes number of faults. Based on this failure information, it has been derived the failure and the system failure data. The tool have been used (SRTpro) software reliability tool professional to extract the results by the tester of project (Park and Baik,2015).

The approach mentioned in chapter 3 will be applied on project-related datasets Parking garage automation as extracting the results. There will be also a copy of data sets in the appendices.

The data of the black box model reliability has been compared it with the data of the proposed model results, and the additive model results.

Finally, Unlike the black box model and the additive model, the proposed model can be distinguished through a real applied result that reliability simulation due to operational profile change. Figure 4.3 shows the Procedures of Experiment.

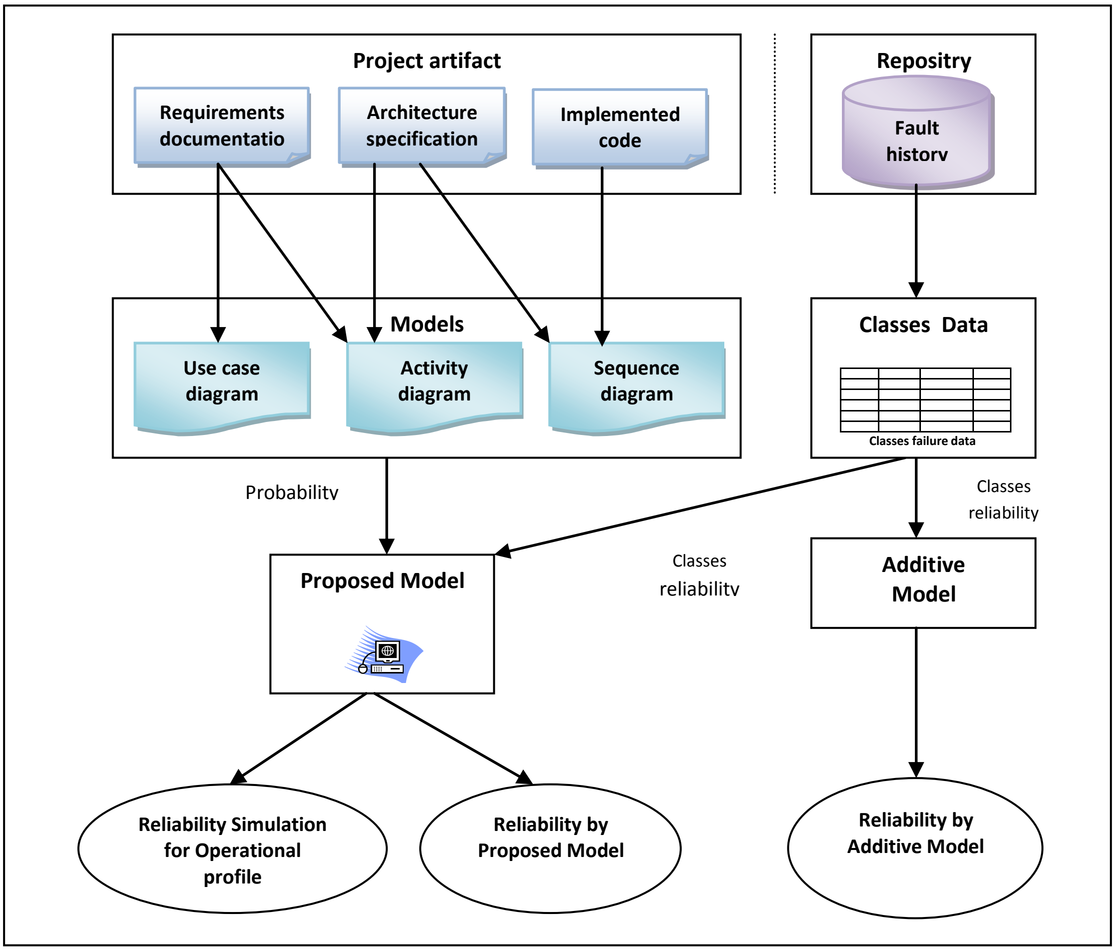


Figure 4.3: Procedures of Experiment.

4.4 The results of the experiment

The proposed model can be derived the system reliability by classes failure data derived from the design model and repository which that producible from early phase artifacts. The comparison of these data with the data derived from the additive model and the black box model to confirm accurately how can the system reliability can be predicted, as seen in Table 4.

Table 4 : Experimental results

Approach	Yamada	Additive Model	Proposed Model
Model Type	Black box model	White box model	White box model
System Reliability	**0.9832**	**0.9571**	**0.9658**
Difference	**-**	**2.61%**	**1.74%**

4.4.1 Black box model (Yamada)

The analysis of the failure data derived from the software repository, can see cumulative failure count an S-shaped curve (black box). The system reliability result derived through the Yamada S-shaped model is **0.9832**

4.4.2 Additive model

The additive model derives the failure intensity simply with the sub-system (classes) failure intensity. The sum of these component failure rates is 0.429 and the entire system reliability according to the following:

assuming, the tests was run ten times (0.429/10 = 0.0429), then system reliability is (1 – 0.0429 = 0.9571).

Table 4.1 : Classes Failure

Class Name	FI
Camera Operator	0.0861
Elevator Operator	0.0587
Sensor Operator	0.0116
Status Display	0.0168
Controller	0.0662
Authorization	0.0120
Account	0.0396
Reservation	0.003
Customers	0.0054
Garage	0.0200
Car	0.0041
PGAfirst	0.1055
TOTAL	**0.429**

4.4.3 Proposed model

The reliability of the project with the proposed approach as explained in chapter 3, by analyzing the operational profile of the system by the session. It is clear that the ratio of the use of each actor to the system, as well as the proportion of the use of each actor to use case, which is known as association probability. By analyzing the session for 100 users, the possibilities in the tables below were determined. Tables 4.2 and 4.3 show actors probability and association probability, to derive system use case.

Table 4.2 : Probability of actor usage

Actor	**Times utilize system**	**P actor**
Registered Customer	53	0.53
Unregistered Customer	20	0.2
System Admin	7	0.07
Elevator Keypad	4	0.04
Elevator Display	4	0.04
Elevator Camera	4	0.04
Spot Sensors	5	0.05
Exit Camera	1	0.01
Timer	2	0.02

Table 4.3 : Association probability

Actor	**Times**	**Use Case**	***P* association (x,y)**	***P* uc (x)**
Registered Customer	20	U1	0.377	0.53*0.377*100 = 19.981
Registered Customer	15	U2	0.283	0.53*0.283*100 = 14.999
Registered Customer	10	U3	0.189	0.53*0.189*100 = 10.017
Registered Customer	5	U4	0.094	0.53*0.094*100 = 4.982
Registered Customer	1	U8	0.019	0.53*0.019*100 = 1.007

Registered Customer	2	U9	0.038	0.53*0.038*100 = 2.014
Unregistered Customer	3	U2	0.15	0.20*0.15*100 = 3
Unregistered Customer	17	U5	0.85	0.20*0.85*100 = 17
System Admin	7	U6	1	0.07*1*100=7
Elevator Keypad	4	U2	1	0.04*1*100=4
Elevator Display	4	U2	1	0.04*1*100=4
Elevator Camera	4	U2	1	0.04*1*100=4
Spot Sensors	5	U2	1	0.05*1*100=5
Exit Camera	1	U2	1	0.01*1*100=1
Timer	2	U13	1	0.02*1*100=2
Registered Customer	-	U2 - > U10	0.283	0.53*0.283*1*100 = 14.999
Registered Customer	-	U3 - > U10	0.189	0.53*0.189*1*100 = 10.017
Registered Customer	-	U4 - > U10	0.094	0.53*0.094*1*100 = 4.982
Registered Customer	-	U4 - > U7	0.094	0.53*0.094*1*100 =4.982
Registered Customer	-	U1 - > U10	0.377	0.53*0.377*1*100 = 19.981
Registered Customer	-	U8 - > U10	0.019	0.53*0.019*1*100 = 1.007
Unregistered Customer	-	U2 - > U10	0.15	0.20*0.15*1*100 = 3
System Admin	-	U6 - > U10	1	0.07*1*1*100=7
System Admin	-	U6 - > U11	0.5	0.07*1*0.6*100 = 4.2
System Admin	-	U6 - > U12	0.5	0.07*1*0.4*100 = 2.8
Total				165.968

Table 4.4 shows each use case's working probability

Table 4.4 : Use case usage of system

Use Case	System Use Case	P norm (x)
U1	Reserve	(19.981/165.968)*100 = 13.039
U2	Park	(35.999/165.968)*100 = 21.690
U3	Manage Account	(10.017/165.968)*100 = 6.035
U4	View Reservation	(4.982/165.968)*100 = 3.001
U5	Register	(17/165.968)*100 = 10.242
U6	Manage Garage	(7/165.968)*100 = 4.218
U7	Edit Reservation	(4.982/165.968)*100 = 3.001
U8	Register Vehicle	(1.007/165.968)*100 = 0.607
U9	Edit Vehicle	(2.014/165.968)*100 = 1.213
U10	Authenticate User	(59.979/165.968)*100 = 36.139
U11	Set Prices	(4.2/165.968)*100 = 2.530
U12	Inspect Usage History	(2.8/165.968)*100 = 1.867
U13	Monthly Billing	(2/165.968)*100 = 1.205

Calculating each use case reliability using the DTMC obtained in step 5 (see Table 4.7). Where, the value of classes failure rate was calculated of similar classes in other similar systems and the cyclomatic complexity number (CCN) is used to obtain the value complexity of method. Tables 4.5 and 4.6 indicate the calculation method respectively, and for further details see chapter 3 stage 4.

Table 4.5 : Expected Failure Rate Methods for Use case 1

Class	**Class Failure Rate (F)**	**Method**	**Complexity Weight Value (W)**	**Expected Failure Of Each Method (θ)**
Reservation	0.003	Make reservation	4	0.012
		Available_ reservations	3	0.009
		Specific_ date and time	2	0.006
		Set reservation	2	0.006

Table 4.6 : Class Failure Rate for Use case 1

Method	**Expected Failure Of Each Method (θ)**	**Busy period**	**Method Failure Rate In The Scenario**	**Results**
Make reservation	0.012	2	1-(1-0.012)^2	0.0238
Available_ reservations	0.009	2	1-(1-0.009)^2	0.0179
Specific_ date and time	0.006	2	1-(1-0.006)^2	0.0119
Set reservation	0.006	1	1-(1-0.006)^1	0.006

The probability of transition *Pi,j* between the modules or methods, *Ni* and *Nj* be :

according to the probability of transition between methods during the examination of 100 cases, thus :

Transition from (Make reservation) to (Available_reservations) was (100) times, so $P_{1,2} = 1$ and transition from (Available_reservations) to (Specific_ date and time) was (100) times, so $P_{2,3} = 1$, $P_{3,4} = 1.00$

And calculate the reliability of each transition are :

R $P_{1,2}$= 1*0.9762 , **R** $P_{2,3}$= 1*0.9821 , **R** $P_{3,4}$= 1.00*0.9881

Table 4.7 is the same as the transition matrix, and to avoid confusion we will refer to it matrix W. Thus, the matrix W represents scenario reliability for use case 1 and also transition matrix.

Table 4.7 : Scenario Reliability for Use case 1

U1 .	**C**	**F**	**1**	**2**	**3**	**4**
C	1	0	0	0	0	0
F	0	1	0	0	0	0
1. Make reservation	0	0.0238	0	0.9762	0	0
2. Available_ reservations	0	0.0179	0	0	0.9821	0
3. Specific_ date and time	0	0.0119	0	0	0	0.9881
4. Set reservation	0.994	0.006	0	0	0	0

If we derived the probability of control being transferred to complete by using DTMC, it can be shown that (Cheung,1980).

$$W = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0.0238 & 0 & 0.9762 & 0 & 0 \\ 0 & 0.0179 & 0 & 0 & 0.9821 & 0 \\ 0 & 0.0119 & 0 & 0 & 0 & 0.9881 \\ 0.994 & 0.0060 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Then extract the matrix Q which represents the following :

$$Q = \begin{pmatrix} 0 & 0.9762 & 0 & 0 \\ 0 & 0 & 0.9821 & 0 \\ 0 & 0 & 0 & 0.9881 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

To solve matrix Q we apply the equation 4.1

R = S(1,n) Rn ....(4.1) (Cheung,1980).

Where , S= $[(\mathrm{I} - \mathrm{Q})^{-1}]$, and I is Identity Matrix

R = S(1,4) R4

(1)*(0.994) = 0.994.

The result 0.994 that reached by this method, we derived the entire use case reliability is displayed in Table 4.7.

The system reliability derived with the proposed model is 96.58, which is within of the reliability result derived from black box model. Table 4.8 shows the results

Table 4.8 : System Reliability

	P	**R**	**P*R**
Reserve	13.039	0.994	12.960
Park	21.690	0.9252	20.067
Manage Account	6.035	0.9011	5.438
View Reservation	3.001	0.0059	2.793
Register	10.242	0.9000	9.217
Manage Garage	4.218	0.8113	3.422
Edit Reservation	3.001	0.8823	2.647
Register Vehicle	0.607	0.9655	0.586
Edit Vehicle	1.213	0.8622	1.045
Authenticate User	36.139	0.9300	33.609
Set Prices	2.530	0.8012	2.027
Inspect Usage History	1.867	0.8500	1.586
Monthly Billing	1.205	0.9210	1.109
	SYSTEM RELIABILITY		**96.58**

Figure 4.4 shows the graphical diagram for system reliability.

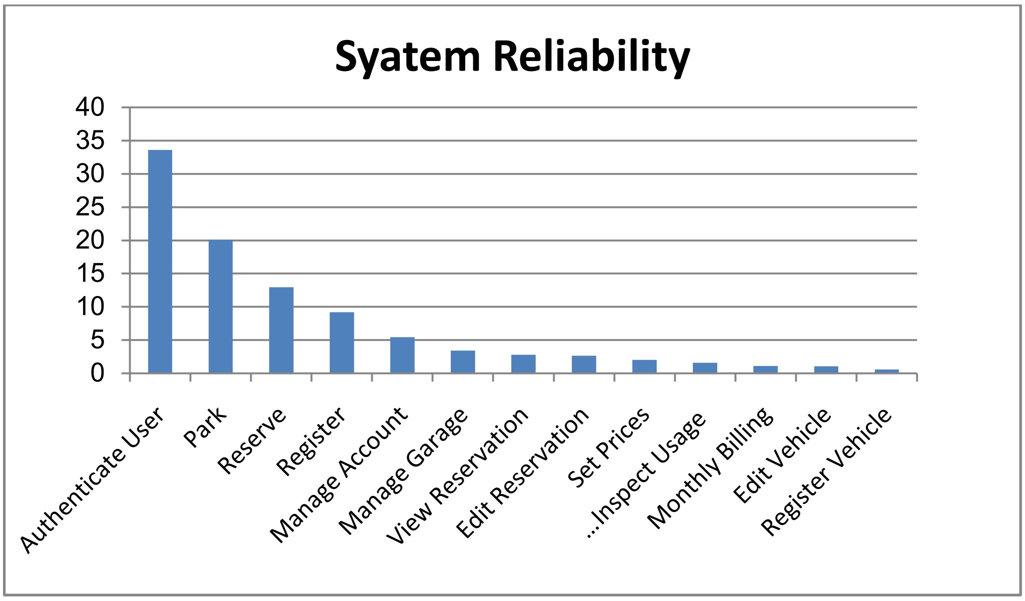


Figure 4.4: Graphical diagram for system reliability.

In general, The proposed model calculate the entire system reliability by deriving the reliability and probability of each system usage level, system activity level, and component interaction level. When, developing the early phase of the system, is hard to accurately predict the operational profile. So, the approach can run a simulation to derive the change of the system reliability due to these types of profile changes.

CHAPTER 5

5.1 Conclusions and future research

The first chapter presents a summary of the study and demonstrates its aspects, objectives, limitations, and the adopted methodology. The second chapter includes a review of relevant prior studies, definitions, software reliability, and as well as adding the interrelation among reliability models and displays the concepts related to white and black boxes. Furthermore, it demonstrates the concept of operational profile and its impact on measuring the reliability, which was ignored in many previous studies. This concept is a fundamental prop in this study. The third chapter represents approach supports early prediction of software reliability, as it is the main task. This approach consists of six main steps by which prediction of software reliability. The fourth chapter lies in the case study to evaluate the approach applicability. Parking garage automation, Through the experiment, the prediction of the approach was evaluated by comparison with existing models the Yamada S-shaped black box model and the additive model. The experimental results show that the proposed method can simulate reliability changes that occur due to operational profile change.

We have encountered some difficulties in terms of shortage of information about the system and its components in an early stage of development. This problem led to difficulty in discovery the source of available information at designing time, which means that it requires understanding the system behavior first. It also caused trouble in applying appropriate mathematical equations, as the study is concerning software engineering. Thus, it necessitates deeper research for the basics of statistics and mathematics in order to reach precise results.

This work paves the way for more research. The obtained evaluation results indicate that this method will provide prediction of software reliability in the context of early stages of its development. It will also concentrate the future research in finding out hybrid methodology of integrating the information from different sources. Introducing hierarchical method will have another scope in research about software reliability. Accuracy of predictions also needs improvement through sensitivity analysis.

Generally, Software-related environment changes rapidly in unpredictable manner. Therefore, reliability of software has to be predicted through the operational profile effect. In design of the early stage of the system, the prediction of operational profile accuracy is difficult. This, in turn, leads to changes in the operational appearance of the test and the operational stage. The proposed method can run an simulate to extract the system reliability change due to these types of changes in operational profile.

References :

1- Khan, H.H. and Malik, M.N., 2017. Software Standards and Software Failures: A Review With the Perspective of Varying Situational Contexts. IEEE Access, 5, pp.17501-17513.

2- Lyu, M.R., 1996. Handbook of software reliability engineering.

3- Alrmuny, D., 2014, April. A Comparative Study of Test Coverage-Based Software Reliability Growth Models. In Information Technology: New Generations (ITNG), 2014 11th International Conference on (pp. 255-259). IEEE.

4- Krajcuskova, Z., 2007, April. Software reliability models. In Radioelektronika, 2007. 17th International Conference (pp. 1-4). IEEE.

5- Everett, W.W., 1999. Software component reliability analysis. In Application-Specific Systems and Software Engineering and Technology, 1999. ASSET'99. Proceedings. 1999 IEEE Symposium on (pp. 204-211). IEEE.

6- Krka, I., Edwards, G., Cheung, L., Golubchik, L. and Medvidovic, N., 2009. A Comprehensive Exploration of Challenges in Architecture-Based Reliability Estimation, Architecting Dependable Systems VI.

7- Blischke, W.R. and Murthy, D.P., 2011. Reliability: modeling, prediction, and optimization (Vol. 767). John Wiley & Sons.

8- Parnas, D.L., 1975, April. The influence of software structure on reliability. In ACM SIGPLAN Notices (Vol. 10, No. 6, pp. 358-362). ACM.

9- Hamlet, D., 1992. Are we testing for true reliability?. IEEE software, 9(4), pp.21-27.

10- Tausworthe, R.C. and Lyu, M.R., 1996. A generalized technique for simulating software reliability. IEEE Software, 13(2), pp.77-88.

11- Yang, M.C. and Chao, A., 1995. Reliability-estimation and stopping-rules for software testing, based on repeated appearances of bugs. IEEE Transactions on Reliability, 44(2), pp.315-321.

12- Dimov, A., Chandran, S.K. and Punnekkat, S., 2010, June. How do we collect data for software reliability estimation?. In Proceedings of the 11th

International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing on International Conference on Computer Systems and Technologies (pp. 155-160). ACM.

13- Chen, M.H., Horgan, J.R., Mathur, A.P. and Rego, V.J., 1992. A time/structure based model for estimating software reliability. Purdue University, SERC-TR-117-P.

14- Murphy, B. and Gent, T., 1995. Measuring system and software reliability using an automated data collection process. Quality and reliability engineering international, 11(5), pp.341-353.

15- Mannhart, A., Bilgic, A. and Bertsche, B., 2007, January. Modeling expert judgment for reliability prediction-comparison of methods. In Reliability and Maintainability Symposium, 2007. RAMS'07. Annual (pp. 1-6). IEEE.

16- Goseva-Popstojanova, K. and Trivedi, K., 2000. Architecture based software reliability.

17- Shanmugam, L. and Florence, L., 2012. An overview of software reliability models. International Journal of Advanced Research in Computer Science and Software Engineering, 2(10).

18- Avizienis, A., Laprie, J.C., Randell, B. and Landwehr, C., 2004. Basic concepts and taxonomy of dependable and secure computing. IEEE transactions on dependable and secure computing, 1(1), pp.11-33.

19- Musa, J.D., 1993. Operational profiles in software-reliability engineering. IEEE software, 10(2), pp.14-32.

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

21- Gokhale, S.S., 2007. Architecture-based software reliability analysis: Overview and limitations. IEEE Transactions on dependable and secure computing, 4(1).

22- Brosch, F., Koziolek, H., Buhnova, B. and Reussner, R., 2012. Architecture-based reliability prediction with the palladio component model. IEEE Transactions on Software Engineering, 38(6), pp.1319-1339.

23- Distefano, S. and Puliafito, A., 2009. Dependability evaluation with dynamic reliability block diagrams and dynamic fault trees. IEEE Transactions on Dependable and Secure Computing, 6(1), pp.4-17.

24- Cortellessa, V., Singh, H. and Cukic, B., 2002, July. Early reliability assessment of UML based software models. In Proceedings of the 3rd international workshop on Software and performance (pp. 302-309). ACM.

25- Rodrigues, G.N., Rosenblum, D.S. and Uchitel, S., 2005, January. Using Scenarios to Predict the Reliability of Concurrent Component-Based Software Systems. In FASE (Vol. 5, pp. 111-126).

26- Gokhale, S.S. and Trivedi, K.S., 2002. Reliability prediction and sensitivity analysis based on software architecture. In Software Reliability Engineering, 2002. ISSRE 2003. Proceedings. 13th International Symposium on (pp. 64-75). IEEE.

27- Reussner, R.H., Schmidt, H.W. and Poernomo, I.H., 2003. Reliability prediction for component-based software architectures. Journal of systems and software, 66(3), pp.241-252.

28- Everett, W.W., 1999. Software component reliability analysis. In Application-Specific Systems and Software Engineering and Technology, 1999. ASSET'99. Proceedings. 1999 IEEE Symposium on (pp. 204-211). IEEE

29- Karanta, I., 2006. Methods and problems of software reliability estimation. VTT WP, 63, p.57.

30- Kashyap, E. and Rana, A., 2015, December. A Comparative Study of S-shape and Concave Software Reliability Growth Models. In Computational Intelligence and Communication Networks (CICN), 2015 International Conference on (pp. 1452-1455). IEEE.

31- Aggarwal, G. and Gupta, V.K., 2014. Software Reliability Growth Model. International Journal of Advanced Research in Computer Science and Software Engineering, 4(1).

32- Mohd, R. and Nazir, M., 2012. Software reliability growth models: Overview and applications. In Journal of Emerging Trends in Computing and Information Sciences VOL.

33- Shooman, M.L., 1976, October. Structural models for software reliability prediction. In Proceedings of the 2nd international conference on Software engineering (pp. 268-280). IEEE Computer Society Press.

34- Goševa-Popstojanova, K. and Trivedi, K.S., 2001. Architecture-based approach to reliability assessment of software systems. Performance Evaluation, 45(2), pp.179-204.

35- An, J. and Zhu, J., 2010, June. Software reliability modeling with integrated test coverage. In Secure Software Integration and Reliability Improvement (SSIRI), 2010 Fourth International Conference on (pp. 106-112). IEEE.

36- Gokhale, S.S. and Trivedi, K.S., 1999. A time/structure based software reliability model. Annals of Software Engineering, 8(1-4), pp.85-121.

37- Edwards, M., Wasserman, E., Hassan, A. and Antialon, J., System Specification and Design: Parking Garage Automation. Interaction, 50, p.50.

38- Park, J. and Baik, J., 2015. Improving software reliability prediction through multi-criteria based dynamic model selection and combination. Journal of Systems and Software, 101, pp.236-244.

39- Cheung, R.C., 1980. A user-oriented software reliability model. IEEE transactions on Software Engineering, (2), pp.118-125.

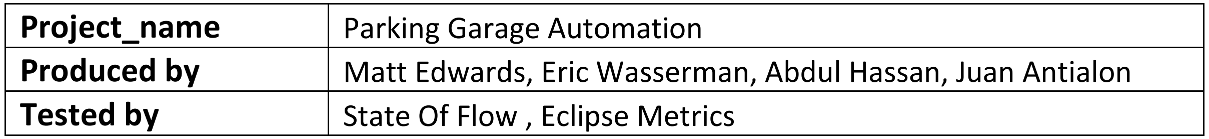
40- Singh, H., Cortellessa, V., Cukic, B., Gunel, E. and Bharadwaj, V., 2001, November. A bayesian approach to reliability prediction and assessment of component based systems. In Software Reliability Engineering, 2001. ISSRE 2001. Proceedings. 12th International Symposium on (pp. 12-21). IEEE.

41- Yacoub, S.M., Cukic, B. and Ammar, H.H., 1999. Scenario-based reliability analysis of component-based software. In Software Reliability Engineering, 1999. Proceedings. 10th International Symposium on (pp. 22-31). IEEE.

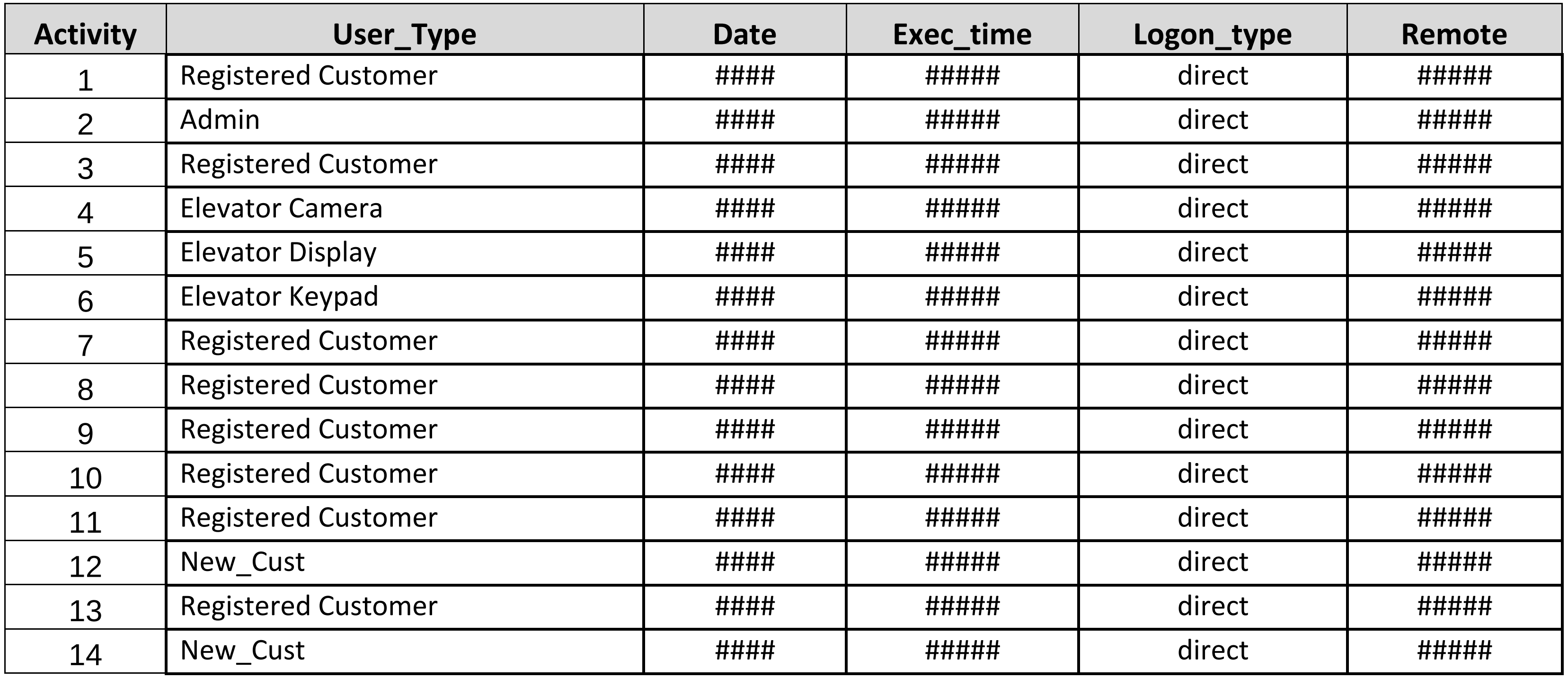
42- Bell, D., 2003. UML basics: An introduction to the Unified Modeling Language. The Rational Edge.

Appendix A

Dataset for using operational profile in general



Project_name	Parking Garage Automation
Produced by	Matt Edwards, Eric Wasserman, Abdul Hassan, Juan Antialon
Tested by	State Of Flow , Eclipse Metrics



Activity	User_Type	Date	Exec_time	Logon_type	Remote
1	Registered Customer	####	#####	direct	#####
2	Admin	####	#####	direct	#####
3	Registered Customer	####	#####	direct	#####
4	Elevator Camera	####	#####	direct	#####
5	Elevator Display	####	#####	direct	#####
6	Elevator Keypad	####	#####	direct	#####
7	Registered Customer	####	#####	direct	#####
8	Registered Customer	####	#####	direct	#####
9	Registered Customer	####	#####	direct	#####
10	Registered Customer	####	#####	direct	#####
11	Registered Customer	####	#####	direct	#####
12	New_Cust	####	#####	direct	#####
13	Registered Customer	####	#####	direct	#####
14	New_Cust	####	#####	direct	#####

Activity	User_Type	Date	Exec_time	Logon_type	Remote
15	Admin	####	#####	direct	#####
16	New_Cust	####	#####	direct	#####
17	New_Cust	####	#####	direct	#####
18	Registered Customer	####	#####	direct	#####
19	Registered Customer	####	#####	direct	#####
20	Registered Customer	####	#####	direct	#####
21	Registered Customer	####	#####	direct	#####
22	New_Cust	####	#####	direct	#####
23	Exit Camera	####	#####	direct	#####
24	New_Cust	####	#####	direct	#####
25	Registered Customer	####	#####	direct	#####
26	Registered Customer	####	#####	direct	#####
27	Registered Customer	####	#####	direct	#####
28	New_Cust	####	#####	direct	#####
29	Registered Customer	####	#####	direct	#####
30	Registered Customer	####	#####	direct	#####
31	Registered Customer	####	#####	direct	#####
32	New_Cust	####	#####	direct	#####
33	Spot Sensor	####	#####	direct	#####
34	Timer	####	#####	direct	#####
35	New_Cust	####	#####	direct	#####
36	Registered Customer	####	#####	direct	#####

Activity	User_Type	Date	Exec_time	Logon_type	Remote
37	New_Cust	####	#####	direct	#####
38	New_Cust	####	#####	direct	#####
39	Registered Customer	####	#####	direct	#####
40	Elevator Camera	####	#####	direct	#####
41	Elevator Display	####	#####	direct	#####
42	Elevator Keypad	####	#####	direct	#####
43	Registered Customer	####	#####	direct	#####
44	New_Cust	####	#####	direct	#####
45	Elevator Camera	####	#####	direct	#####
46	Elevator Display	####	#####	direct	#####
47	Elevator Keypad	####	#####	direct	#####
48	Spot Sensor	####	#####	direct	#####
49	Registered Customer	####	#####	direct	#####
50	Registered Customer	####	#####	direct	#####
51	Registered Customer	####	#####	direct	#####
52	Spot Sensor	####	#####	direct	#####
53	Registered Customer	####	#####	direct	#####
54	Registered Customer	####	#####	direct	#####
55	Admin	####	#####	direct	#####
56	Registered Customer	####	#####	direct	#####
57	Registered Customer	####	#####	direct	#####
58	Registered Customer	####	#####	direct	#####

Activity	User_Type	Date	Exec_time	Logon_type	Remote
59	Registered Customer	####	#####	direct	#####
60	Registered Customer	####	#####	direct	#####
61	Admin	####	#####	direct	#####
62	New_Cust	####	#####	direct	#####
63	Registered Customer	####	#####	direct	#####
64	New_Cust	####	#####	direct	#####
65	Spot Sensor	####	#####	direct	#####
66	Registered Customer	####	#####	direct	#####
67	Registered Customer	####	#####	direct	#####
68	Registered Customer	####	#####	direct	#####
69	Registered Customer	####	#####	direct	#####
70	Registered Customer	####	#####	direct	#####
71	Registered Customer	####	#####	direct	#####
72	Registered Customer	####	#####	direct	#####
73	Admin	####	#####	direct	#####
74	Admin	####	#####	direct	#####
75	Elevator Camera	####	#####	direct	#####
76	Elevator Display	####	#####	direct	#####
77	Elevator Keypad	####	#####	direct	#####
78	Spot Sensor	####	#####	direct	#####
79	New_Cust	####	#####	direct	#####
80	Registered Customer	####	#####	direct	#####

Activity	User_Type	Date	Exec_time	Logon_type	Remote
81	Registered Customer	####	#####	direct	#####
82	Registered Customer	####	#####	direct	#####
83	Timer	####	#####	direct	#####
84	Registered Customer	####	#####	direct	#####
85	Registered Customer	####	#####	direct	#####
86	Registered Customer	####	#####	direct	#####
87	Registered Customer	####	#####	direct	#####
88	Registered Customer	####	#####	direct	#####
89	Registered Customer	####	#####	direct	#####
90	New_Cust	####	#####	direct	#####
91	Registered Customer	####	#####	direct	#####
92	New_Cust	####	#####	direct	#####
93	New_Cust	####	#####	direct	#####
94	Admin	####	#####	direct	#####
95	Registered Customer	####	#####	direct	#####
96	Registered Customer	####	#####	direct	#####
97	Registered Customer	####	#####	direct	#####
98	New_Cust	####	#####	direct	#####
99	Registered Customer	####	#####	direct	#####
100	New_Cust	####	#####	direct	#####

Appendix B

Dataset for using operational profile in details

User_Type	Method	Times
Registered Customer	Make_reservation	20
	Open_elevator	15
	get_info	10
	getreservation	5
	Add_ vehicle	1
	Update_ vehicle	2
		total = 53
New_Cust	pay_walk in	3
	Create_account	17
		total = 20
Admin	Update_acc	1
	Update_gar	1
	setprice	3
	Calculate_delay	2
		total = 7
Timer	Notice_time	2
		total = 2
Elevator Camera	Identify_ license plate	4
		total = 4
Elevator Display	Open_elevator	4

User_Type	Method	Times
		total = 4
Elevator Keypad	Open_elevator	4
		total = 4
Exit Camera	Display _exit camera	1
		total = 1
Spot Sensor	setspot occupied	5
		total = 5

Appendix C

Dataset for methods in the project

Index	
Short Name	**Full Name**
CC	Cyclomatic Complexity
LOCm	Lines of Code in Method
NLS	Number of Locals in Scope
NOL	Number of Levels
NOP	Number of Parameters
BB	Busy Period
NOS	Number of Statements

N	Methods		Metrics Values			
	Method _number	**Method _name**	**CC**	**NLS**	**NOL**	**BP**
1	1.1	Make_reservation	4	4	3	2
2	1.2	Available_reservations	3	2	2	2
3	1.3	Specific_date and time	2	2	2	2
4	1.4	setreservation	2	2	2	1
5	2.1	Open_elevator	2	2	2	2
6	2.2	Identify_ license plate	3	2	2	2
7	2.3	getplate	3	2	1	1

N	Methods		Metrics Values			
	Method _number	Method _name	CC	NLS	NOL	BP
8	2.4	Assign _parking spot	2	2	1	2
9	2.5	Display_ parking spot	2	2	1	2
10	2.6	Display_ camera elevator	2	2	1	2
11	2.7	Display_ sensor assign	2	2	1	2
12	2.8	setspot occupied	1	2	1	2
13	2.9	pay_walk in	3	3	1	3
14	2.10	Display _exit camera	2	1	1	2
15	2.11	Spot _sensor free	2	1	1	1
16	2.12	setspot unoccupied	1	1	2	2
17	3.1	get_info	1	1	1	1
18	3.2	Display_info	2	1	1	1
19	3.3	Valid_info	2	1	1	1
20	3.4	setinfo	1	1	1	1
21	4.1	getreservation	1	1	1	2
22	5.1	Create_account	3	2	1	2
23	5.2	valid_info	2	1	1	1
24	5.3	setinfo	1	1	2	1
25	6.1	Add_manger	3	1	2	1
26	6.2	Update_acc	3	2	2	1
27	6.3	Add_gar	2	2	2	1
28	6.4	Update_gar	1	2	2	1
29	6.5	setprice	4	1	2	2
30	6.6	Calculate_delay	5	2	2	2

N	Methods		Metrics Values			
	Method _number	Method _name	CC	NLS	NOL	BP
31	6.7	Del_ customer	2	2	2	2
32	6.8	valid_info	1	1	2	1
33	7.1	Update_ reservation	2	1	2	2
34	7.2	Valid_info	1	1	2	1
35	7.3	Del_ reservation	2	1	2	1
36	8.1	Add_ vehicle	2	1	2	2
37	8.2	Valid_info	1	1	1	1
38	8.3	setinfo	1	1	1	1
39	9.1	Display_info	1	1	1	1
40	9.2	Update_ vehicle	2	1	1	2
41	9.3	Del_ vehicle	2	1	1	2
42	9.4	Valid_data	1	1	1	1
43	9.5	setvehicle	1	1	1	2
44	10.1	Fill_data	2	1	1	1
45	10.2	Valid_data	1	1	1	1
46	10.3	getdata	2	1	1	1
47	10.4	Display_info	2	1	1	1
48	11.1	Gar_location	2	1	1	1
49	11.2	Confirm_reservation rate	2	1	1	1
50	11.3	Penalty fees	3	3	1	2
51	11.4	setprice	1	1	1	2
52	11.5	getinfo	1	1	1	1
53	11.6	Valid_info	1	1	1	1

N	Methods		Metrics Values			
	Method _number	Method _name	CC	NLS	NOL	BP
54	11.7	setinfo	1	1	1	1
55	12.1	Search_criteria	2	1	1	2
56	12.2	gethistory	3	1	1	2
57	12.3	vaild_info	2	1	1	1
58	12.4	setinfo	2	1	1	1
59	13.1	Notice_time	3	1	1	2
60	13.2	getcustomers	3	1	1	2
61	13.3	setinfo	2	1	1	1
62	13.4	Send_Email	2	1	1	2

Appendix D

Dataset for methods transition and related

N	Meth _Num	Use Case associated	Class associated	Failure Intensity of class	State	Meth _Num	Execution Times
1	1.1	Reserve	Reservation	0.003	non	1.1	0
2	1.1	Reserve	Reservation	0.003	To	1.2	100
3	1.1	Reserve	Reservation	0.003	To	1.3	0
4	1.1	Reserve	Reservation	0.003	To	1.4	0
5	1.2	Reserve	Reservation	0.003	To	1.1	0
6	1.2	Reserve	Reservation	0.003	non	1.2	0
7	1.2	Reserve	Reservation	0.003	To	1.3	100
8	1.2	Reserve	Reservation	0.003	To	1.4	0
9	1.3	Reserve	Reservation	0.003	To	1.1	0
10	1.3	Reserve	Reservation	0.003	To	1.2	0
11	1.3	Reserve	Reservation	0.003	non	1.3	0
12	1.3	Reserve	Reservation	0.003	To	1.4	100
13	1.4	Reserve	Reservation	0.003	-	-	-
14	2.1	Park	Elevator Operator	0.0587	non	2.1	0
15	2.1	Park	Elevator Operator	0.0587	To	2.2	86
16	2.1	Park	Elevator Operator	0.0587	To	2.3	0
17	2.1	Park	Elevator Operator	0.0587	To	2.4	0
18	2.1	Park	Elevator Operator	0.0587	To	2.5	0
19	2.1	Park	Elevator Operator	0.0587	To	2.6	0
20	2.1	Park	Elevator Operator	0.0587	To	2.7	0
21	2.1	Park	Elevator Operator	0.0587	To	2.8	0

N	Meth _Num	Use Case associated	Class associated	Failure Intensity of class	State	Meth _Num	Execution Times
22	2.1	Park	Elevator Operator	0.0587	To	2.9	14
23	2.1	Park	Elevator Operator	0.0587	To	2.10	0
24	2.1	Park	Elevator Operator	0.0587	To	2.11	0
25	2.1	Park	Elevator Operator	0.0587	To	2.12	0
26	2.2	Park	Camera Operator	0.0861	To	2.1	0
27	2.2	Park	Camera Operator	0.0861	non	2.2	0
28	2.2	Park	Camera Operator	0.0861	To	2.3	100
29	2.2	Park	Camera Operator	0.0861	To	2.4	0
30	2.2	Park	Camera Operator	0.0861	To	2.5	0
31	2.2	Park	Camera Operator	0.0861	To	2.6	0
32	2.2	Park	Camera Operator	0.0861	To	2.7	0
33	2.2	Park	Camera Operator	0.0861	To	2.8	0
34	2.2	Park	Camera Operator	0.0861	To	2.9	0
35	2.2	Park	Camera Operator	0.0861	To	2.10	0
36	2.2	Park	Camera Operator	0.0861	To	2.11	0
37	2.2	Park	Camera Operator	0.0861	To	2.12	0
38	2.3	Park	Camera Operator	0.0861	To	2.1	0
39	2.3	Park	Camera Operator	0.0861	To	2.2	0
40	2.3	Park	Camera Operator	0.0861	non	2.3	0
41	2.3	Park	Camera Operator	0.0861	To	2.4	100
42	2.3	Park	Camera Operator	0.0861	To	2.5	0
43	2.3	Park	Camera Operator	0.0861	To	2.6	0
44	2.3	Park	Camera Operator	0.0861	To	2.7	0
45	2.3	Park	Camera Operator	0.0861	To	2.8	0
46	2.3	Park	Camera Operator	0.0861	To	2.9	0

N	Meth _Num	Use Case associated	Class associated	Failure Intensity of class	State	Meth _Num	Execution Times
47	2.3	Park	Camera Operator	0.0861	To	2.10	0
48	2.3	Park	Camera Operator	0.0861	To	2.11	0
49	2.3	Park	Camera Operator	0.0861	To	2.12	0
50	2.4	Park	Sensor Operator	0.0116	To	2.1	0
51	2.4	Park	Sensor Operator	0.0116	To	2.2	0
52	2.4	Park	Sensor Operator	0.0116	To	2.3	0
53	2.4	Park	Sensor Operator	0.0116	non	2.4	0
54	2.4	Park	Sensor Operator	0.0116	To	2.5	100
55	2.4	Park	Sensor Operator	0.0116	To	2.6	0
56	2.4	Park	Sensor Operator	0.0116	To	2.7	0
57	2.4	Park	Sensor Operator	0.0116	To	2.8	0
58	2.4	Park	Sensor Operator	0.0116	To	2.9	0
59	2.4	Park	Sensor Operator	0.0116	To	2.10	0
60	2.4	Park	Sensor Operator	0.0116	To	2.11	0
61	2.4	Park	Sensor Operator	0.0116	To	2.12	0
62	2.5	Park	Sensor Operator	0.0116	To	2.1	0
63	2.5	Park	Sensor Operator	0.0116	To	2.2	0
64	2.5	Park	Sensor Operator	0.0116	To	2.3	0
65	2.5	Park	Sensor Operator	0.0116	To	2.4	0
66	2.5	Park	Sensor Operator	0.0116	non	2.5	0
67	2.5	Park	Sensor Operator	0.0116	To	2.6	100
68	2.5	Park	Sensor Operator	0.0116	To	2.7	0
69	2.5	Park	Sensor Operator	0.0116	To	2.8	0
70	2.5	Park	Sensor Operator	0.0116	To	2.9	0
71	2.5	Park	Sensor Operator	0.0116	To	2.10	0

N	Meth _Num	Use Case associated	Class associated	Failure Intensity of class	State	Meth _Num	Execution Times
72	2.5	Park	Sensor Operator	0.0116	To	2.11	0
73	2.5	Park	Sensor Operator	0.0116	To	2.12	0
74	2.6	Park	Camera Operator	0.0861	To	2.1	0
75	2.6	Park	Camera Operator	0.0861	To	2.2	0
76	2.6	Park	Camera Operator	0.0861	To	2.3	0
77	2.6	Park	Camera Operator	0.0861	To	2.4	0
78	2.6	Park	Camera Operator	0.0861	To	2.5	0
79	2.6	Park	Camera Operator	0.0861	non	2.6	0
80	2.6	Park	Camera Operator	0.0861	To	2.7	100
81	2.6	Park	Camera Operator	0.0861	To	2.8	0
82	2.6	Park	Camera Operator	0.0861	To	2.9	0
83	2.6	Park	Camera Operator	0.0861	To	2.10	0
84	2.6	Park	Camera Operator	0.0861	To	2.11	0
85	2.6	Park	Camera Operator	0.0861	To	2.12	0
86	2.7	Park	Sensor Operator	0.0116	To	2.1	0
87	2.7	Park	Sensor Operator	0.0116	To	2.2	0
88	2.7	Park	Sensor Operator	0.0116	To	2.3	0
89	2.7	Park	Sensor Operator	0.0116	To	2.4	0
90	2.7	Park	Sensor Operator	0.0116	To	2.5	0
91	2.7	Park	Sensor Operator	0.0116	To	2.6	0
92	2.7	Park	Sensor Operator	0.0116	non	2.7	0
93	2.7	Park	Sensor Operator	0.0116	To	2.8	100
94	2.7	Park	Sensor Operator	0.0116	To	2.9	0
95	2.7	Park	Sensor Operator	0.0116	To	2.10	0
96	2.7	Park	Sensor Operator	0.0116	To	2.11	0

N	Meth _Num	Use Case associated	Class associated	Failure Intensity of class	State	Meth _Num	Execution Times
97	2.7	Park	Sensor Operator	0.0116	To	2.12	0
98	2.8	Park	Sensor Operator	0.0116	-	-	-
99	2.9	Park	Elevator Operator	0.0587	To	2.1	0
100	2.9	Park	Elevator Operator	0.0587	To	2.2	100
101	2.9	Park	Elevator Operator	0.0587	To	2.3	0
102	2.9	Park	Elevator Operator	0.0587	To	2.4	0
103	2.9	Park	Elevator Operator	0.0587	To	2.5	0
104	2.9	Park	Elevator Operator	0.0587	To	2.6	0
105	2.9	Park	Elevator Operator	0.0587	To	2.7	0
106	2.9	Park	Elevator Operator	0.0587	To	2.8	0
107	2.9	Park	Elevator Operator	0.0587	non	2.9	0
108	2.9	Park	Elevator Operator	0.0587	To	2.10	0
109	2.9	Park	Elevator Operator	0.0587	To	2.11	0
110	2.9	Park	Elevator Operator	0.0587	To	2.12	0
111	2.10	Park	Camera Operator	0.0861	To	2.1	0
112	2.10	Park	Camera Operator	0.0861	To	2.2	0
113	2.10	Park	Camera Operator	0.0861	To	2.3	0
114	2.10	Park	Camera Operator	0.0861	To	2.4	0
115	2.10	Park	Camera Operator	0.0861	To	2.5	0
116	2.10	Park	Camera Operator	0.0861	To	2.6	0
117	2.10	Park	Camera Operator	0.0861	To	2.7	0
118	2.10	Park	Camera Operator	0.0861	To	2.8	0
119	2.10	Park	Camera Operator	0.0861	To	2.9	0
120	2.10	Park	Camera Operator	0.0861	non	2.10	0
121	2.10	Park	Camera Operator	0.0861	To	2.11	100

N	Meth _Num	Use Case associated	Class associated	Failure Intensity of class	State	Meth _Num	Execution Times
122	2.10	Park	Camera Operator	0.0861	To	2.12	0
123	2.11	Park	Sensor Operator	0.0116	To	2.1	0
124	2.11	Park	Sensor Operator	0.0116	To	2.2	0
125	2.11	Park	Sensor Operator	0.0116	To	2.3	0
126	2.11	Park	Sensor Operator	0.0116	To	2.4	0
127	2.11	Park	Sensor Operator	0.0116	To	2.5	0
128	2.11	Park	Sensor Operator	0.0116	To	2.6	0
129	2.11	Park	Sensor Operator	0.0116	To	2.7	0
130	2.11	Park	Sensor Operator	0.0116	To	2.8	0
131	2.11	Park	Sensor Operator	0.0116	To	2.9	0
132	2.11	Park	Sensor Operator	0.0116	To	2.10	0
133	2.11	Park	Sensor Operator	0.0116	non	2.11	0
134	2.11	Park	Sensor Operator	0.0116	To	2.12	100
135	2.12	Park	Sensor Operator	0.0116	-	-	-
136	3.1	Manage Account	Account	0.0396	non	3.1	0
137	3.1	Manage Account	Account	0.0396	To	3.2	100
138	3.1	Manage Account	Account	0.0396	To	3.3	0
139	3.1	Manage Account	Account	0.0396	To	3.4	0
140	3.2	Manage Account	Account	0.0396	To	3.1	0
141	3.2	Manage Account	Account	0.0396	non	3.2	0
142	3.2	Manage Account	Account	0.0396	To	3.3	100
143	3.2	Manage Account	Account	0.0396	To	3.4	0
144	3.3	Manage Account	Account	0.0396	To	3.1	0
145	3.3	Manage Account	Account	0.0396	To	3.2	0
146	3.3	Manage Account	Account	0.0396	To	3.3	0

N	Meth _Num	Use Case associated	Class associated	Failure Intensity of class	State	Meth _Num	Execution Times
147	3.3	Manage Account	Account	0.0396	To	3.4	100
148	3.4	Manage Account	Account	0.0396	-	-	-
149	4.1	View Reservation	Reservation	0.003	-	-	100
150	5.1	Register	customers	0.0054	non	5.1	0
151	5.1	Register	customers	0.0054	To	5.2	100
152	5.1	Register	customers	0.0054	To	5.3	0
153	5.2	Register	customers	0.0054	To	5.1	0
154	5.2	Register	customers	0.0054	non	5.2	0
155	5.2	Register	customers	0.0054	To	5.3	100
156	5.3	Register	customers	0.0054	-	-	-
157	6.1	Manage Garage	Garage	0.02	non	6.1	0
158	6.1	Manage Garage	Garage	0.02	To	6.2	0
159	6.1	Manage Garage	Garage	0.02	To	6.3	0
160	6.1	Manage Garage	Garage	0.02	To	6.4	0
161	6.1	Manage Garage	Garage	0.02	To	6.5	0
162	6.1	Manage Garage	Garage	0.02	To	6.6	0
163	6.1	Manage Garage	Garage	0.02	To	6.7	100
164	6.2	Manage Garage	Garage	0.02	To	6.1	0
165	6.2	Manage Garage	Garage	0.02	non	6.2	0
166	6.2	Manage Garage	Garage	0.02	To	6.3	0
167	6.2	Manage Garage	Garage	0.02	To	6.4	0
168	6.2	Manage Garage	Garage	0.02	To	6.5	0
169	6.2	Manage Garage	Garage	0.02	To	6.6	0
170	6.2	Manage Garage	Garage	0.02	To	6.7	100
171	6.3	Manage Garage	Garage	0.02	To	6.1	0

N	Meth _Num	Use Case associated	Class associated	Failure Intensity of class	State	Meth _Num	Execution Times
172	6.3	Manage Garage	Garage	0.02	To	6.2	0
173	6.3	Manage Garage	Garage	0.02	non	6.3	0
174	6.3	Manage Garage	Garage	0.02	To	6.4	0
175	6.3	Manage Garage	Garage	0.02	To	6.5	0
176	6.3	Manage Garage	Garage	0.02	To	6.6	0
177	6.3	Manage Garage	Garage	0.02	To	6.7	100
178	6.4	Manage Garage	Garage	0.02	To	6.1	0
179	6.4	Manage Garage	Garage	0.02	To	6.2	0
180	6.4	Manage Garage	Garage	0.02	To	6.3	0
181	6.4	Manage Garage	Garage	0.02	non	6.4	0
182	6.4	Manage Garage	Garage	0.02	To	6.5	0
183	6.4	Manage Garage	Garage	0.02	To	6.6	0
184	6.4	Manage Garage	Garage	0.02	To	6.7	100
185	6.5	Manage Garage	Garage	0.02	To	6.1	0
186	6.5	Manage Garage	Garage	0.02	To	6.2	0
187	6.5	Manage Garage	Garage	0.02	To	6.3	0
188	6.5	Manage Garage	Garage	0.02	To	6.4	0
189	6.5	Manage Garage	Garage	0.02	non	6.5	0
190	6.5	Manage Garage	Garage	0.02	To	6.6	0
191	6.5	Manage Garage	Garage	0.02	To	6.7	100
192	6.6	Manage Garage	Garage	0.02	To	6.1	0
193	6.6	Manage Garage	Garage	0.02	To	6.2	0
194	6.6	Manage Garage	Garage	0.02	To	6.3	0
195	6.6	Manage Garage	Garage	0.02	To	6.4	0
196	6.6	Manage Garage	Garage	0.02	To	6.5	0

N	Meth _Num	Use Case associated	Class associated	Failure Intensity of class	State	Meth _Num	Execution Times
197	6.6	Manage Garage	Garage	0.02	non	6.6	0
198	6.6	Manage Garage	Garage	0.02	To	6.7	100
199	6.7	Manage Garage	Garage	0.02	-	-	-
200	7.1	Edit Reservation	Reservation	0.003	non	7.1	0
201	7.1	Edit Reservation	Reservation	0.003	To	7.2	0
202	7.1	Edit Reservation	Reservation	0.003	To	7.3	100
203	7.2	Edit Reservation	Reservation	0.003	To	7.1	0
204	7.2	Edit Reservation	Reservation	0.003	To	7.2	0
205	7.2	Edit Reservation	Reservation	0.003	To	7.3	100
206	7.3	Edit Reservation	Reservation	0.003	-	-	-
207	8.1	Register Vehicle	Car	0.0041	non	8.1	0
208	8.1	Register Vehicle	Car	0.0041	To	8.2	100
209	8.1	Register Vehicle	Car	0.0041	To	8.3	0
210	8.2	Register Vehicle	Car	0.0041	To	8.1	0
211	8.2	Register Vehicle	Car	0.0041	non	8.2	0
212	8.2	Register Vehicle	Car	0.0041	To	8.3	100
213	8.3	Register Vehicle	Car	0.0041	-	-	-
214	9.1	Edit Vehicle	Car	0.0041	non	9.1	0
215	9.1	Edit Vehicle	Car	0.0041	To	9.2	90
216	9.1	Edit Vehicle	Car	0.0041	To	9.3	10
217	9.1	Edit Vehicle	Car	0.0041	To	9.4	0
218	9.1	Edit Vehicle	Car	0.0041	To	9.5	0
219	9.2	Edit Vehicle	Car	0.0041	To	9.1	0
220	9.2	Edit Vehicle	Car	0.0041	non	9.2	0
221	9.2	Edit Vehicle	Car	0.0041	To	9.3	0

N	Meth _Num	Use Case associated	Class associated	Failure Intensity of class	State	Meth _Num	Execution Times
222	9.2	Edit Vehicle	Car	0.0041	To	9.4	100
223	9.2	Edit Vehicle	Car	0.0041	To	9.5	0
224	9.3	Edit Vehicle	Car	0.0041	To	9.1	0
225	9.3	Edit Vehicle	Car	0.0041	To	9.2	0
226	9.3	Edit Vehicle	Car	0.0041	non	9.3	0
227	9.3	Edit Vehicle	Car	0.0041	To	9.4	100
228	9.3	Edit Vehicle	Car	0.0041	To	9.5	0
229	9.4	Edit Vehicle	Car	0.0041	To	9.1	0
230	9.4	Edit Vehicle	Car	0.0041	To	9.2	0
231	9.4	Edit Vehicle	Car	0.0041	To	9.3	0
232	9.4	Edit Vehicle	Car	0.0041	non	9.4	0
233	9.4	Edit Vehicle	Car	0.0041	To	9.5	100
234	9.5	Edit Vehicle	Car	0.0041	-	-	-
235	10.1	Authenticate User	Authorization	0.012	non	10.1	0
236	10.1	Authenticate User	Authorization	0.012	To	10.2	100
237	10.1	Authenticate User	Authorization	0.012	To	10.3	0
238	10.1	Authenticate User	Authorization	0.012	To	10.4	0
239	10.2	Authenticate User	Authorization	0.012	To	10.1	0
240	10.2	Authenticate User	Authorization	0.012	non	10.2	0
241	10.2	Authenticate User	Authorization	0.012	To	10.3	100
242	10.2	Authenticate User	Authorization	0.012	To	10.4	0
243	10.3	Authenticate User	Authorization	0.012	To	10.1	0
244	10.3	Authenticate User	Authorization	0.012	To	10.2	0
245	10.3	Authenticate User	Authorization	0.012	To	10.3	0
246	10.3	Authenticate User	Authorization	0.012	To	10.4	100

N	Meth _Num	Use Case associated	Class associated	Failure Intensity of class	State	Meth _Num	Execution Times
247	10.4	Authenticate User	Authorization	0.012	-	-	-
248	11.1	Set Prices	Garage	0.02	non	11.1	0
249	11.1	Set Prices	Garage	0.02	To	11.2	0
250	11.1	Set Prices	Garage	0.02	To	11.3	0
251	11.1	Set Prices	Garage	0.02	To	11.4	100
252	11.1	Set Prices	Garage	0.02	To	11.5	0
253	11.1	Set Prices	Garage	0.02	To	11.6	0
254	11.1	Set Prices	Garage	0.02	To	11.7	0
255	11.2	Set Prices	Garage	0.02	To	11.1	0
256	11.2	Set Prices	Garage	0.02	non	11.2	0
257	11.2	Set Prices	Garage	0.02	To	11.3	0
258	11.2	Set Prices	Garage	0.02	To	11.4	0
259	11.2	Set Prices	Garage	0.02	To	11.5	100
260	11.2	Set Prices	Garage	0.02	To	11.6	0
261	11.2	Set Prices	Garage	0.02	To	11.7	0
262	11.3	Set Prices	Garage	0.02	To	11.1	0
263	11.3	Set Prices	Garage	0.02	To	11.2	0
264	11.3	Set Prices	Garage	0.02	non	11.3	0
265	11.3	Set Prices	Garage	0.02	To	11.4	0
266	11.3	Set Prices	Garage	0.02	To	11.5	100
267	11.3	Set Prices	Garage	0.02	To	11.6	0
268	11.3	Set Prices	Garage	0.02	To	11.7	0
269	11.4	Set Prices	Garage	0.02	To	11.1	0
270	11.4	Set Prices	Garage	0.02	To	11.2	0
271	11.4	Set Prices	Garage	0.02	To	11.3	0

N	Meth _Num	Use Case associated	Class associated	Failure Intensity of class	State	Meth _Num	Execution Times
272	11.4	Set Prices	Garage	0.02	non	11.4	0
273	11.4	Set Prices	Garage	0.02	To	11.5	0
274	11.4	Set Prices	Garage	0.02	To	11.6	100
275	11.4	Set Prices	Garage	0.02	To	11.7	0
276	11.5	Set Prices	Garage	0.02	To	11.1	0
277	11.5	Set Prices	Garage	0.02	To	11.2	0
278	11.5	Set Prices	Garage	0.02	To	11.3	0
279	11.5	Set Prices	Garage	0.02	To	11.4	0
280	11.5	Set Prices	Garage	0.02	To	11.5	0
281	11.5	Set Prices	Garage	0.02	To	11.6	100
282	11.5	Set Prices	Garage	0.02	To	11.7	0
283	11.6	Set Prices	Garage	0.02	To	11.1	0
284	11.6	Set Prices	Garage	0.02	To	11.2	0
285	11.6	Set Prices	Garage	0.02	To	11.3	0
286	11.6	Set Prices	Garage	0.02	To	11.4	0
287	11.6	Set Prices	Garage	0.02	To	11.5	0
288	11.6	Set Prices	Garage	0.02	To	11.6	0
289	11.6	Set Prices	Garage	0.02	To	11.7	100
290	11.7	Set Prices	Garage	0.02	-	-	-
291	12.1	Inspect Usage History	Garage	0.02	non	12.1	0
292	12.1	Inspect Usage History	Garage	0.02	To	12.2	100
293	12.1	Inspect Usage History	Garage	0.02	To	12.3	0
294	12.1	Inspect Usage History	Garage	0.02	To	12.4	0
295	12.2	Inspect Usage History	Garage	0.02	To	12.1	0
296	12.2	Inspect Usage History	Garage	0.02	To	12.2	0

N	Meth _Num	Use Case associated	Class associated	Failure Intensity of class	State	Meth _Num	Execution Times
297	12.2	Inspect Usage History	Garage	0.02	To	12.3	100
298	12.2	Inspect Usage History	Garage	0.02	To	12.4	0
299	12.3	Inspect Usage History	Garage	0.02	To	12.1	0
300	12.3	Inspect Usage History	Garage	0.02	To	12.2	0
301	12.3	Inspect Usage History	Garage	0.02	To	12.3	0
302	12.3	Inspect Usage History	Garage	0.02	To	12.4	100
303	12.4	Inspect Usage History	Garage	0.02	-	-	-
304	13.1	Monthly Billing	customers	0.0054	non	13.1	0
305	13.1	Monthly Billing	customers	0.0054	To	13.2	100
306	13.1	Monthly Billing	customers	0.0054	To	13.3	0
307	13.1	Monthly Billing	customers	0.0054	To	13.4	0
308	13.2	Monthly Billing	customers	0.0054	To	13.1	0
309	13.2	Monthly Billing	customers	0.0054	non	13.2	0
310	13.2	Monthly Billing	customers	0.0054	To	13.3	100
311	13.2	Monthly Billing	customers	0.0054	To	13.4	0
312	13.3	Monthly Billing	customers	0.0054	To	13.1	0
313	13.3	Monthly Billing	customers	0.0054	To	13.2	0
314	13.3	Monthly Billing	customers	0.0054	non	13.3	0
315	13.3	Monthly Billing	customers	0.0054	To	13.4	100
316	13.4	Monthly Billing	customers	0.0054	-	-	-

الخلاصة

على الرغم من أن دراسات موثوقية البرمجيات قد جذبت قدرا كبيرا من الاهتمام, حيث تتناول هذه الأطروحة احد التحديات الهامة في تحليل موثوقية البرمجيات. نهج المربع الأبيض في تحليل موثوقية البرمجيات , كما توفر معلومات مفيدة لاتخاذ قرار أكثر دقة من خلال تحديد أجزاء معينة من التطبيق والتي لن تسبب أي فشل للنظام. حيث تعقب الأجزاء المحتمل أن تكون غير موثوقة في المراحل المبكرة أمر صعب بسبب عدم توفر كود قابل للتنفيذ.

في الجزء النظري، يشير نهج المربع الأبيض في تحليل موثوقية البرمجيات والذي يدعم التنبؤ في مرحلة التصميم، والنهج المقترح يقسم عملية تحليل الموثوقية إلى ست مراحل, حيث استخدام أجزاء التصاميم مثل مخطط حالة الاستخدام والنشاط، مخطط تسلسل أو تتبع المكونات. كما يتنبأ نهج موثوقية النظام من خلال تقدير شدة فشل مستوى الأسلوب أو الدالة من خلال عدد الاستدعاءات وهي الفترة التي تكون الدالة مشغولة في أداء مهمة معينة وقيمة وزن التعقيد للدالة. ومن خلال حساب احتمال نقل النشاط إلى حالة كاملة وهي حالة النجاح باستخدام سلسلة ماركوف. وكذلك ، من الممكن لمحاكاة كيفية تغيير موثوقية النظام عندما يتم تغيير التشكيل الجانبي التشغيلي للنظام.

ومن الناحية العملية، أجريت دراسة تجريبية لتقييم تطبيق النهج المقترح. وأظهرت نتائج الدراسة أن التقنية يمكن التنبؤ موثوقية البرمجيات بشكل أكثر دقة ويحاكي التغييرات الشخصي.

التنبؤ بموثوقية البرمجيات باستخدام لغة النمذجة المؤحدة

قدمت من قبل :

عبدالغني يونس عبدالغني

تحت إشراف

د. محمد أخليف

قدمت هذه الرسالة استكمالا لمتطلبات الحصول على درجة الماجستير في علوم الحاسوب

جامعة بنغازي
كلية تقنية المعلومات

يوليو 2018