Ranking of Software Engineering Metrics by Fuzzy-Based Matrix Methodology

A major Dissertation Submitted in partial fulfilment of the requirement for the award of the degree of

MASTER OF TECHNOLOGY

In

COMPUTER SCIENCE AND ENGINEERING

Submitted by:

SURENDRA TYAGI

Roll No.:- 2K11/CSE/26

Under the guidance of:

Dr. KAPIL SHARMA

Delhi Technological University

Department of Computer Engineering

Delhi Technological University

Bawana Road Delhi-110042

2013-2014

Ranking of Software Engineering Metrics by Fuzzy-Based Matrix Methodology

A major Dissertation Submitted in partial fulfilment

of the requirement for the award of the degree of

MASTER OF TECHNOLOGY

In

COMPUTER SCIENCE AND ENGINEERING

Submitted by:

SURENDRA TYAGI

Roll No.:- 2K11/CSE/26

Under the guidance of:

Dr. KAPIL SHARMA

Delhi Technological University

Department of Computer Engineering

Delhi Technological University

Bawana Road Delhi-110042

2013-2014

ABSTRACT

In present day a lot of software's are being developed day by day. To check reliability of software is a big issue. For this software engineering metrics is formed and ranked by different methods. In present thesis work we proposed a method for ranking of software engineering metrics based on expert's opinions elicitation and fuzzy-based matrix methodology. The proposed methodology has ability to translate the vague and imprecise data concerned with the problem of ranking of software engineering metrics, and the ambiguity and uncertainty occurring at the time of expert decision making to remove the complexity of formulation of the intention and the control function. The matrices provide themselves to mechanical manipulations and are helpful for evaluating and developing systems functions which match with the purpose of research work. This research work is based on software engineering metrics acknowledged in an earlier study conducted by Lawrence Livermore National Laboratory. A set of ranking criteria were recognized. After that software engineering metrics are ranked in ascending series using experts' opinion in according to the values of Permanent function on their criteria matrix. The proposed methodology has also been compared with other known methodologies.

The use of fuzzy set theory improves the decision-making procedure by considering the vagueness and ambiguity prevalent in real-world system. We also found that the use of triangular fuzzy numbers made data collection, calculation, and interpretation of results easier for experts.

DECLARATION

I hereby declare that the thesis entitled "**Ranking of Software Engineering Metrics by Fuzzy-Based Matrix Methodology"** which is being submitted to the **Delhi Technological University**, in partial fulfillment of the requirements for the award of degree of **Master of Technology in Computer Science and Engineering** is an authentic work carried out by me. The material contained in this thesis has not been submitted to any university or institution for the award of any degree.

SURENDRA TYAGI
Master of Technology
(Computer Science and Engineering)
College Roll No. 2K11/CSE/26
Department of Computer Engineering
Delhi Technological University,
Delhi.

CERTIFICATE

Date:

This is to certify that the thesis entitled **"Ranking of Software Engineering Metrics by Fuzzy-Based Matrix Methodology"** submitted by **SURENDRA TYAGI (Roll Number: 2K11/CSE/26),** in partial fulfillment of the requirements for the award of degree of Master of Technology in Computer Science and Engineering, is an authentic work carried out by him under my guidance.

The content embodied in this thesis has not been submitted to any institution or University for any degree or diploma to the best of my knowledge and belief.

Dr. Kapil Sharma

Associate Professor

Department of Computer Engineering

Delhi Technological University

ACKNOWLEDGEMENT

With due regards, I hereby take this opportunity to acknowledge a lot of people who have supported me with their words and deeds in completion of my research work as part of this course of Master of Technology in Computer Science and Engineering.

To start with I would like to thank the almighty god and Shri Shri 108 Baba Narayan Das for being with me in each and every step of my life. Next, I thank my parents and family, for their encouragement and persistent support.

I would like to express my deepest sense of gratitude and indebtedness to my guide and motivator, **Dr. Kapil Sharma**, Associate Professor, Department of Computer Engineering, Delhi Technological University for his valuable guidance and support in all the phases from conceptualization to final completion of the thesis. It is his immense support that enabled me to complete my thesis in time.

I wish to convey my sincere gratitude to Prof. Daya Gupta, Prof. O.P. Verma (DTU), Prof. Ashok de (NIT Patna), Dr. Vishal Bhatnagar (Assosite professor AIACTR) and all the faculties of Computer Engineering Department, DTU who have been a source of inspiration and continuously enlightened me during my thesis.

I humbly extend my grateful appreciation to my social welfare organisation SEWA BHARTI, Sh. Babu lal jee (Secretary Uttari Vibhag Sewa Bharti), Balak Ram (LNJP hospital) my friends Devanand Meena, Dr. Ashutosh Kumar(AIIMS), Saket mani Trivedi(Director KCRR), Satish Tyagi, Dileep Tyagi, Rajesh Kumar, Rohit, Vikrant, Anjana, Saurabh,Zeehshan, My elder sister Dr. Amita Sharma, my social work colleagues Sh. Yogendra Rana , Sh. Mohit singh , Venu Sahai whose moral support made this thesis possible.

Last but not the least; I would like to thank all the people directly and indirectly involved in successfully completion of this thesis.

SURENDRA TYAGI

Roll No. 2K11/CSE/26

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREBARATION AND SYMBOLS:

FD	Fault Density
MTTF	Mean Time To Failure
ROCOF	Rate Of Failure Occurrence
ANOVA	Analysis Of Variance
AHP	Analytic Hierarchy Process
LLNL	Lawrence Livermore National Laboratory
IEEE	Institute Of Electrical And Electrical Engineering
GUI	Graphical User Interface
CC	Cyclomatic Complexity
SDC	System Design Complexity
RC	Requirements Compliance
CH	Cohesion
M	Medium
H	High
L	Low
R(p)	Reliability Metrics
$f_A(x)$	Membership function
c_t	Criteria
w_t	Weight
R_{it}	Rating Assigned To Software Engineering Metric
$U_M(F_i)$	Left Utility Value
$U_G(F_i)$	Right Utility Value
$U_T(F_i)$	Total Utility Value
$\lambda(p)$	The failure intensity
BBN	Bayesian Belief Network

Chapter One: **INTRODUCTION**

1.1 INTRODUCTION

In modern society, software has become a very significant element in all types of systems. New softwares are being developed everyday. Many of them are useful, while majority of them are not matching to the desired satisfaction of the user.Developing a software that is trustworthy is invaluable, but what is the cost of developing software that is substandard? A famaous tagedy in 1999, NASA lost the Mars Lander because of an error made by software development team who provided software to calculate distances in Metric and English units but failed to design software to make right conversions between the two. NASA lost valuable time, money and pride on a simple error that must have been detected prior to deployment of the Mars Lander. (NASA, 2014)

NASA's damage was significant, but a larger mis-happening occurred in 1991 when a Patriot Scud missile used during the Persian Gulf War failed to sense an incoming scud missile. The Patriot Scud, which was earlier failed due to its accuracy, had a short rounding error in the timer (approximately 0.000000095 seconds for every second of time the Patriot Missile was in use). The timer, which was needed for computing distances of incoming scud missiles, added an error of approximately 0.34 seconds over 100 hours of operation,this time was sufficient to fail to sense an incoming scud missile. The Patriot Scud failure had taken the lives of 28 American soldiers. (Arnold, 2014)

Software crashes occur every day, most of the time however, the crashes are not as expensive as the Mars Lander Failure and Patriot Missile Failure, but are an bother none the less. Software failures is the most crucial issue that stops the work and affect right manner functioning of the whole system. hence, it is very essential and vital to remove as many probable problems in software as possible. The software development work has become more and more time-wasting and costly because of the complexity of software systems. In this time, the requirement for extremely reliable software system is ever increasing. How to increase the quality of the software systems and decrease the expenditure to an adequate level becomes a major concern of present software industry. Methods of applying reliability and cost models to the software development process are extremely needed. (Pham & Zhang, 1999).

Most vital and dynamic feature of software is its reliability. Generally, the reliability of a software system is a measure of how well it provides the services expected of it by its users but a useful proper definition of reliability is much difficult to explain. Software reliability metrics such as 'mean time between failures may be used.

Reliability is a dynamic system feature, which is a function of the number of software failures. A software failure is an execution event where the software does not behaves in an expected way. This is not the same as a software fault, which is a static program characteristic. Software faults cause software failures when the faulty code is run with a specific set of inputs. Faults do not always show themselves as failures so the reliability is subjected to on how the software is used. It is not possible to make a single, universal statement of the software reliability. (DewSoft, 2014)

Software faults are not just program defects. Unexpected behavior can occur in circumstances where the software conforms to its requirements themselves are complete. Omissions in software documentations can also lead to unexpected behavior, although the software may not contain defects.

There is a intricate connection between experimental reliability and the number of hidden errors. It points out that not all software errors have an equal probability of occuring. Removing software faults from parts of the system, which are seldom used, makes little difference to the perceived reliability. Their work recommended that, for the products studied, removing 60% of product defects would lead to only a 3% improvement in reliability.

Reliability is to be subject to on how the software is used, so it cannot be quantified absolutely. Different users uses a program in different ways so mistakes, which affect the reliability of the system for one user, may never manifest themselves under a different method of working. Reliability can only be accurately specified if the normal software operational profile is also detailed.

As reliability is subjected to the probability of an error occuring in functioning use, a program may have known errors but may still never be seen select an mistaken input; the program always appears to be reliable. In addition, skilled operators may ‘work around’ identified software errors and consciously escape using features, which they recognize to be mistaken. Repairing the errors in these features may make no practical change to the reliability as supposed by these users.

Software reliability may be defined as the probability that software will not cause a failure of a system for a specifics time under specifics conditions, and is one of the greatest vital appearances of quality. Maximum of the software reliability models that were made to specify the probability of software failure are based on software failure observations prepared for the duration of test or operation(MR, 1995.)(Musa, A, & Okumoto, 1987)(Ramamoorthy & Bastani, 1982). The conventional software reliability models may not put on in definite cases where it may not be possible to detect an suitable number of failures. It may also be the situation that some companies or research institutions designate to quantity other software engineering metrics like complexity and fault density (FD). Thus, software reliability may have to be evaluated *a posteriori* from existing sets of software engineering metrics.

To explain this issue, it may be supposed that the product characteristics and the operational environment are two issues that contribute for defining software reliability. Further, the project features, as the type of application, functional size, etc., and the development features, type as the developer’s ability, project inexpensive, stiffness of timetable, methods, tools, and languages needed for the development of the product, define product characteristics.

1.2 MATHEMATICS OF RELIABILITY:

Reliability of a product quantified the probability of without failure working of that product for a given time duration. Unreliability of any product occurs because of failures or mainly of errors in the system. As software does not "wear out" or "age" as a mechanical or an electronic system does, the unreliability of software is mainly due to errors or design bugs in the software. It is commonly supposed that with the present time of technology, it is not possible to identify and remove the entire error in a big software system (mostly before supply). Consequently, a software system is possible to have some faults in it.

Reliability is a probabilistic quantity that adopts that the manifestation of failure of software is a random phenomenon. i.e., if it is well defined the total workable time of a software system as a variable, this is a random variable that may assume dissimilar values in dissimilar calls of the software. This randomness of the failure incidences is essential for reliability modeling. Here, by *randomness* all that is intended is that the failure cannot be predicted exactly. This hypothesis will normally hold for bigger systems, but may not hold for little programs that have bugs (in which situation, one might be able to prediction the errors). Therefore, reliability modeling is more significant for larger systems. It is recommended that it must be functional to systems bigger than 5000 LOC, because such systems will give enough data points to do statistical examination.

Let y be the random variable that denotes the life time of a system. The *failure probability,* F (p), of a system is well defined as the probability that the system will be breakdown by time *p* i.e., the life of the system, y is less than *p*

Equation 1-1

$$F(p)=P(y\leq p)$$

As *F (p)* denotes the failure probability till a given time *p* which changes with time some may denotes functions for *F (p)*. This function is known as the *failure distribution function.* Each functions must have a 0 at time *p= 0* (a system cannot be value of failed earlier time 0) and a value 1 at time p= y all systems must be failed before infinite time). System reliability is the probability that the system has not failed till time *t.* In other words,

Equation 1-2

$$R(p)=1-F(p).$$

If *F (p)* is differentiable, its first derivative *f (p)* is called the *failure density function.* The failure density function shows the instantaneous failure probability at time *p.* Or, the probability that a failure will arise between times *p* and *(p + ~p)* is given by *f (p) ~p.*

These definitions give the failure probability, and reliability, failure density as a function of time at the starting time. i.e., at time *p* = 0 it is forecasting that the probability that the system should be failed by some time *p* is *F (p).* What happen it is observed that by time *p the* system has not failed (after all *F (p)* is only a probability)? That is, as time passes, it is found that a system has not failed by some time *p.* In that situation, at time *p,* we would like to identify the future breakdown probabilities from that time ahead. In other words, it identify the failure probability for a system, specified that the system has not failed by time *p.* This is usually specified for a system by its *hazard rate, z (p),* which is the conditional failure density at time *p,* given that no failure has occurred between 0 and *p.* By this definition, the hazard rate is

Equation 1-3

$$Z\ (p) = f\ (p)/R\ (p)$$

The connection between the hazard rate and reliability is

Equation 1-4

$$R\ (p) = e^{\int_0^t Z(y)dy}$$

The reliability of a system may also be defined as the *mean time to failure (MTTF).* MTTF shows the probable lifetime of the system. From the reliability function, it can be found as:

Equation 1-5

$$MTTF = \int_0^\infty R(y)dy$$

Note that one can find the MTTF from the reliability function but the opposite is not forever true. The reliability function may, however, be found from the MTTF if the failure action is understood to be *Poisson,* that is, *F (p)* has an *exponential distribution.* Exponential distribution is given by

Equation 1-6

$$F(p) = 1 - e^{\gamma_p}$$

Where γ_p the failure is rate and is equal to inverse of MTTF.

The traditional definition of reliability was given earlier. Previous, there are other types in which reliability can be defined. In the preceding definitions, the random variable was taken as the time to next failure or the life of the system. We can describe a different random variable, which signifies the number of failures expert by the system by time *p.* obviously; this number will also be random, because failures are random. If the random variable representing the number of failures till time *p,* it can be define that reliability in another form. If m*(p)* denotes the distribution of the number of failures felt by time *p,* then the *mean value function μ(p)*

Equation 1-7

$$\mu(p) = E\,[M\,(p)]$$

Where *E* is the expectation function and *μ(p)* represents the probable number of failures that will be occurred till time *p.* The function *μ(p)* have a value of 0 at time p = *0* and be a non-decreasing function. *The failure intensity* $\lambda(p)$ of the system is explained as:

Equation 1-8

$$\lambda(t) = \frac{d\mu(t)}{dt} \quad , \quad \lambda = d\mu(p)/dp$$

The failure intensity quantifies the instantaneous medication in the probable number of failures, or the probable number of failures per unit time. The number of failures that happens between p and $p + \Delta(p)$ can be estimates as $\lambda(p)\,\Delta(p)$. For Poisson- like system (where failure probability has an exponential distribution), the probability of more than one time occurring in a little duration $\Delta(p)$is supposed 0. therefore, $\Delta(p)$ shows the probability that a failure will happen in between p and *(p*

$+\Delta(p))$, that is the similar as the probability that time the system does not fail up to time t and at is a failure time $\Delta(t)$after the time. In this way, for these kinds of models, the hazard rate is the similar as the failure intensity function.

1.2.1 Meaning of Time in Reliability models:

There are three general explanations of time for software reliability models: execution time, calendar time, and clock time.

Execution time is the really CPU time, the software runs for the period of its execution.

Calendar time is the ordered time that is used by group.

Clock time is the actual clock time that passes when the software is running (i.e., it contains the time the software waits also consist in it other).

Different models define in different manner definitions, although the maximum generally used are execution time and calendar time. It supposed that execution time models are for superior and more precise than calendar time models, since they more precisely define the "stress" on the software because of processing.

In software, what is called a "failure" is subject to the project, and its exact description is provided by the tester or project manager. For a line, is a misplaced line in the output may be a failure or not? Obviously, it is instance on the project; someone would consider it a failure and others will not. Another example. Determined output is not being generated in a given time duration, is it a failure or not? in a real-time system this may be supposed as a failure, however for an operating system it may not be supposed as a failure. That is there is no clear cut definition of failure, and it depends of project manager or end user to They will decide what will be supposed a failure for reliability purposes. Note that in the example of a misplaced line, a flaw might be registered, and even make right after some time, but its occurring may not be qualified a failure. The failure feature of software is mainly given by two things:

1. The number of errors in the software being evaluated.

2. The profile of operation of the execution.

This is clear, with a large number of fault in code of, anyone will think the software to be a low reliability. i.e., the more errors, the more the probability that the system will be filled more within time *p. i.e.* by the total number of error in the software can be supposed to be a rough guide of its reliability. so *flaws* or *faults per KLOC* are a very commonly used metric for measuring quality. This type of metric is used to compare processes or products to quantify reliability. As far as a metric does not need reliability modeling, which need a reasonable quantity of data gathering and complexity, this metric is broadly used in practice, in spite of its limitations., the reliability models can be used to estimate the faults per KLOC metric more accurately.

The failure of software is also subjected to seriously on the atmosphere in which it is working. It is famous concept that software mostly fails only if some different types of inputs are given. In other ways, if software has faults, merely some particular types of input will generate cause exercise that fault to influence failures. Therefore, how many times these inputs generate failures at the time of execution will decide how many times the software fails. The functioning outline of software confine the probability of different kinds of inputs being provided to the software at the time of its execution. Since explanation of reliability is based on failures that depend on the nature of inputs reliability is clearly dependent on the operational profile of the software. Hence, when we say that the reliability of software is *R (p),* it assumes that this is for some operational profile. If the operational profile changes dramatically, then it will be need to either recomputed *R (p)* or recalculate it. In other words, to measure the reliability of a software system, it must be observed the failures of the software in the operational profile in which it is finally going to execute. Normally, it is supposed that the profile of inputs given during system testing is same as to the inputs the software will understand during operation (i.e., the test cases at the time system testing are with the operational outline of the software). Therefore, the data of system testing is used to model the reliability of the software. (DewSoft, 2014)

Software reliability metrics have, by and large, developed from hardware reliability metrics. Though, hardware metrics cannot be used without amendment due to the dissimillar nature of software and hardware failures.

1.3 SOFTWARE ENGINEERING MATRICS

Once calculated data are gathered they are changed into metrics for use. IEEE defines metric as 'a quantitative measure of the degree to which a system, component, or process possesses a given attribute.' The purpose of software metrics is to identity and manage essential issues that influence software work. Other purpose of software metrics are listed following

To compute the dimensions of the software quantitatively.

To help the stage of complexity exist.

To help the powering of the component by calculating correction coupling.

To help the testing methods.

To guide when to stop testing.

To determine the time of completion of the software.

To approximate required cost of resources and project calender.

Software metrics assist project managers to increase an insight into the productivity of the software development, project, and product. It is possible by bringing collectively excellence and efficiency statistics and then examining and matching these statistics with last averages to identify whether excellence enhancements have taken place. Also, when metrics are used in a consistent manner, it assist in project planning and project management activity. For example, schedule-based resource allotment can be efficiently improved with the help of metrics.

1.3.1 Distinction in Measures, Metrics, and Indicators

Metrics is frequently said interchangeably with measure or measurement. Through, it is crucial to know the dissimilarties between them. Measure may be known as quantitative clue of amount, size, capacity, or dimension of product and process attributes. Measurement is defined as the process of determining the measure. Metrics can be defined as quantitative measures that permit software engineers to recognize the productvity and get better the excellence of software process, project, and product

To comprehend the differentiation, let us take an example. A measure is established which a number of errors are (single data point) sensed in a software part. Measurement is the technique of gathering one or more data points. i.e., measurement is recognized when a more elements are re-evaluate and tested separaty to get together the measure of a more errors in all these components. Metrics are linked with particular measure in some type behaviour. In other ways, metrics are associated to recognition of errors found per re-evlauation or the average number of errors recognised per unit test.

Once measures and metrics have been made, indicators are found. These indicators provide a whole knowledge of the software process, software project, or inter stage product. Indicators also make able software engineers or project managers to control software processes and get for better qualtiy software products, if required. For example, measurement dashboards or key indicators are used to see development and initiate change. Making manage collectively, indicators give pictures of the system's performance.

1.3.1.1 Measured Data

Before data is gathered and used, it is needed to recognize the kind of data related in the software metrics. Make table lists different kinds of data, which are identified in metrics beside with their detail and the probable actions that can be performed on them

1.3.1.1.1 Type of Data Measured

Type of data	**Possible operations**	**Description of data**
Nominal	=,≠	Categories
Ordinal	<, >	Ranking
Interval	+, -	Differences
Ratio	/	Absolute zero

1.3.1.1.1.1 Nominal data:

Data in the program can be identified by putting it in a cluster. This cluster of program can be a database program, application program, or an operating system program. For such kind of data, activites of addition subtraciton type and ordering of values in any order (increasing or decreasing) is not possible. The merely action that can be done is to make certain whether program 'X' is the qual to 'Y'.

1.3.1.1.1.2 Ordinal data:

Data can be ordered with calculation of to the data values. For illustaution, experience in application domain can be given as very low, low, medium, or high. Thus, experience can easily be ordered according to its rating.

1.3.1.1.1.3 Interval data:

Data values can be ordered and considerable gap between them can also be given. For illution, a program with complexity standard 8 is said to as 4 units more complex than a program with complexity standard 4.

1.3.1.1.1.4 Ratio data:

Data values are related to a ratio scale, which holds an exact zero and permits signigicant ratios to be mesured. For illustation, program lines represent in lines of code.

It is perferable to recognise the measurement scale for metrics. For illustation, if metrics values are used to show a model for a software process, then metrics related with the ratio scale may be chosen.

1.3.2 Guidelines for Software Metrics

Although lot of software metrics have been projected till this time, best software metric is the one which is simple to understand, efficient, and effective. For the sake of developing best metrics, software metrics must be validated and categorised effectively. For this, it is vital to make metrics using some given rules, which are following.

1.3.2.1 Easy and computable:

calculation of software metrics must be simple to understand and should consume average quantity of time and effort.

1.3.2.2 Consistent and objective:

Clear-cut data must be generated by software metrics.

1.3.2.3 Consistent in the use of units and dimensions:

Mathematical calculation of the metrics should contain use of dimensions and units in a steady manner.

1.3.2.4 Programming language independent:

Metrics must be made on the basis of the analysis model, design model, or program's structure.

1.3.2.5 High quality:

Efficient and Effective software metrics should lead to a high-excellence software product.

1.3.2.6 Easy to standardize:

Metrics should be easy to adjust according to project needs.

1.3.2.7 Easy to obtain:

Metrics should be made at a reasonable charge.

1.3.2.8 Validation:

Metrics should be sustificate before being used for taking any decisions.

1.3.2.9 Robust:

Metrics should be comparatively insensible to small alterations in process, project, or product.

1.3.2.10 Value:

Value of metrics should increase or decrease according value of the software features they show. For this, the value of metrics must be within a reasonable order. For example, metrics can be in a order of 0 to 5. (Thakur, 2014)

1.4 RELIABILITY METRICS

There are some metrics, which are used to measure software reliability are:

1.4.1 Probability of failure on demand:

This is a quantification of the chances that the system will work in an undesired method when some demand is made on it. It is generally associated with secure complex systems and "continuous" systems whose continuous action is critical. In these systems, a determine of failure occuring is less vital that the possibility that the system will not behave as expected.

1.4.2 Rate of failure occurrence (ROCOF):

This is a measure of the frequency of incidence with which unforeseen behavior is likely to be shown. For illustration, if the ROCOF is 2/100 this tells that 2 failures are probable to happen in each 100 action time units. time units are discussed shortly. This is, possibly, the most generally useful reliability metric.

1.4.3 Mean time to failure (MTTF):

This is a calculation of the time during shown failures. This metric is a similar to a comparable metric used in hardware reliability evaluation where it represents the life time of system element. In software systems, parts do not wear out and, They remain in action after a single failure. hence, mean time to failure is merely helpful in software reliability evaluation when the system is constant and no alternations are being done to it. In this case, it gives a signal of how long the system will be in a action before a failure occurs.

1.4.4 Availability:

This quantify the system is to be available for use. For illustration, an availability of 998/1000 means that in every 1000 time units, the system is probable to be available for 998 of these. This measure is most suitable for systems like telecommunication systems, where the repair or start again at time is important and the loss of working in this interval during time is vital. (Software realiabilty, 2014)

No lone metric is unanimously suitable and the specific metric used should depend on the application domain and the projected usage of the system. For large systems, it may be suitable to use different reliability metrics for different component of the system. All the above features can be quantified explicitly or implicitly using software engineering metrics. Hence, an obvious inference is that 'software engineering metrics determine software reliability' (Li & Smidts, A ranking of software engineering measures based on expert opinion, 2003).

Parastoo and Dehlen (Parastoo & Dehlen, May 2009) discussed the use of metrics for measuring quality. They are presented an general idea of planned metrics in literature and some examples of usage. Ordonez and Haddad(Ordonez & Haddad, April 2008) examined the practices of metrics in software industry and experiences of some related organizations. These experiences show proof of profit and progress in excellence and reliability. Software engineering metrics, used for reliability estimation and assurance, are ranked in terms of their capacity to forecast software reliability. It is very vital to rank software engineering metrics since top-ranked metrics are the expected roots of total set of metrics to find authentic reliability forcasting. The ranking is also important for software industry for better organization and quality manage of software development work and hence to increase the software quality. Ranking of the software engineering metrics on the basis of lots of criteria creates a multi-criteria decision-making problem. The values provided to chosen criteria are frequently qualitatively described or imprecisely measured. The significance of each factor may also change in different requirements and situations. It is easier for a decision maker to describe his/her desired value and the importance of a criterion by using common language. Owing to the vaguely nature of software engineering metrics and the ranking criteria, there is a need to expand a multi-criteria decision-making method based on fuzzy set theory. Fuzzy set theory was developed to address this exact hypothesis, that the key parts in human thinking are not interger value, but linguistic terms or fuzzy set labels (Zadeh, 1965).

The aim of doing this experimental research is to progress the understanding of software engineering metrics that may have power on software reliability and evaluate the importance of their effects. Thus, it needs developing a fuzzy-based matrix methodology to systematically rank the available software engineering metrics with respect to their effect on the forecast of software reliability. This thesis is organized as follows: in Chapter 2, a review of software engineering metrics ranking problem and framework is provided. A decision-making approach based on fuzzy sets and matrix operations is described in Chapter 3. In Chapter 4, a ranking procedure for software engineering metrics is described. The application of the fuzzy sets and matrix operation for ranking of software engineering metrics based on different criteria using an illustrated example is presented in Chapter 5. The results are analyzed in Chapter 6. Chapter 6 also compares the proposed method with the existing methods and Chapter 7 contains the conclusion of the methodology presented in this thesis.

Chapter Two: **REVIEW OF LITERATURE**

Roberts *et al.*(Roberts, Gibson, Fields, & Rainer, 1998)recognized five factors vital to implementing a system development methodology. Understanding the significance of aspects that affects the software metrics, it was also told that there is a need for analytical methodologies that put together both software complexity metrics and different parameters describing the software development environment(Evanco & Lacovara, 1994) . Schneberger(Schneberger, 1997) represented the outcomes of his work of the impacts of distributed computing environments on software maintenance difficulty. Furuyama *et al.*(Furuyama, Yoshio, & Kazuhiko, Fault generation model and mental stress effect analysis., 1994)(Furuyama, Arai, & Lio, Analysis of fault generation caused by stress during software development, 1997) studied criteria such as working stress, development methodologies, etc. They found that various settings of these factors have statistically significant effect on the quality of final software products.

Zhang and Pham (Zhang & Pham, 2000)conducted a survey and found qualitative and quantitative data from software professionals and managers of 13 top companies. The relative weight and analysis of variance (ANOVA) methods were used to examine the known 32 factors affecting software reliability. In this study no exact expert elicitation process was described. The expert biases were not measured and the relative weight method was not acceptable.

Fenton and Neil (Fenton & Neil, 1999) projected a Bayesian Belief Network (BBN) model to forecast software defect density, and Johnson and Yu (Johnson & Yu, Objective software quality assessment., 1999) presented a BBN software quality model, based on BBN technique that finds software reliability by software engineering metrics measurement. These procedures need large amount of data and that's why such procedures cannot be broadly used. Moreover, correctness of the results cannot be ensured.

Analytic Hierarchy Process (AHP) has been used to select software reliability metrics(Li, Lu, & Li, 2006). It occupies large amount of time for calculation and is also hard to score while the number of the criteria increases greater than seven.

Criteria interdependency can put up with losses because of oversimplifying the hierarchy and assessment of the quality for software parts (Sharma, Kumar, & Grover, 2008).

A limited number of expert views applications are got in the software engineering area. Putnam and Fitzsimmons(Putnam & Fitzsimmons, 1979) projected a subjective approximation of the length of a program. Kitchenham *et al.* (Kitchenham, Linkman, & Law, 1997) examined software engineering methods and tools using subject surveys as one of the valuation methods. Dyba (Dyba, 2000) used expert view to recognize and rank the key factors of success in software process enhancement of quality. Wohlin *et al.* (Wohlin, Mayrhauser, Host, & Regnell, 2000) approximated the success of a project using subjective factors. Host and Wohlin (Host & Wohlin, 1998)(Host & Wohlin, 1998) performed attempt evaluation by merging individual evaluations performed by field specialists. Briand *et al.* (Briand, Freimut, & Vollei, 2000) suggested an expert view application to the estimation of the cost effectiveness of examination. Many researchers (Li, et al., 2004)(Singh, Singh, & Singh, 2006) have anticipated methods for ranking of software engineering measures based on specialist view. In these studies, suspicions and partiality in the expert's decisions are purposely reduced. Moreover, the liner additive plans used to aggregate the scores elicited through specialist views are comparatively inflexible, inexact, and also does not think the comparative weights, i.e. interdependencies of software engineering metrics. In case of analysis through AHP it is very hard to be pair wise comparison particularly when a lot of metrics are concerned and thus becomes a somewhat complex problem to solve.

From the wide study of the exists literature, it is found that there is a need to make a unified way that can make suitable imprecision and vagueness occurring at the time of human decision making and will make able to consider all ranking criteria and their relative importance concomitantly in an incorporated approach for ranking of software engineering metrics. consequently, fuzzy-based matrix method (a traditional multi-attribute decision-making computation method) to deal with specialist decisions qualitatively and quantitatively is proposed. The technique is more flexible, correct, and has better understanding and reliability. The decision-making methods, using fuzzy set theory, have slowly got recognition over the last decade and their applications have also become more in different areas. A complete explanation of these applications can be found in (Chang & Chen, 1994)(Wang & Chang, 1995)(Liao, 1996)(Yeh, Deng, & Chang, 2000)(Karsak & Tolga, 2001)(Lau, et al., 2003)(Wang & Lin, 2003)(McIvor, McCloskey, Humphreys, & Maguire, 2004)(Cochran & Chen, 2005)(Garg, Gupta, & Agrawal, 2007)(Khatatnech & Mustafa, 2009)(Bailador & Trivi, 2010)

Chapter Three: **METHODOLOGY ADOPTED**

To minimize complexity of the calculation of objective and constraint functions that is faced when the mathematical programming model is used in a multi attributes decision problem, an initiative has been introduced in this thesis to generate a finite quantitative model based on fuzzy sets and matrix operations for the sake of ranking of software engineering metrics.

The projected fuzzy-based matrix methodology has been made suitable and implemented for ranking of software engineering metrics. In this methodology the use of fuzzy set theory makes suitable the ambiguity and vagueness faced at the time human decision making. The matrices are useful in analyzing expectations and to make the system function and index to meet the purposes.

A short introduction to crucial concepts of fuzzy sets, algebraic operations, triangular fuzzy numbers, linguistic variables, and matrix operations is presented in this chapter.

3.1 FUZZY SETS

Fuzzy set theory, consisting of the fuzziness of data, was given by Zadeh (Zadeh, 1965). It was generated to draw solutions of problems, in which details of activities and observations were vague, unclear, and uncertain. A fuzzy set is a class of objects, with a range of membership degree, where the membership degree is taken value between 0 and 1. A fuzzy subset A of a universal set X is given by a membership function $f_A(x)$ which maps each element x in X to a real number (0, 1). The degree of membership for an element is 1, that is the element is in that set. The degree of membership is 0, that's meaning is that the element is not in that set. In ambiguous cases membership values are given between 0 and 1. The theory also permits mathematical operations such as addition, subtraction, multiplication, and division. Which can be applied on the fuzzy sets (Kaufmann & Gupta, 1988)(Dubois D, Prade H. Fuzzy real algebra: Some results, 1979).

3.2 TRIANGULAR FUZZY NUMBERS

In this study, triangular fuzzy numbers are used as membership functions, related to the elements in a set, as shown in Figure 1. The reason for using a triangular fuzzy number is that it is naturally easy for the decision makers to use and compute. A fuzzy number is a triangular fuzzy number if its membership function can be givn as follows (Kaufmann & Gupta, 1988).

Equation 3-1

$$f_A(x) = \begin{cases} \frac{x-c}{a-c} & c \leq x \leq a \\ \frac{b-x}{b-a} & c \leq x \leq b \\ 0 & otherwise \end{cases}$$

Where a, b, and c are real numbers and $c \leq a \leq b$.

Zadeh's extension principle is used to compute membership functions. In this study, only addition and multiplication are used. Defining two triangular fuzzy numbers A_1 and A_2 by the

.

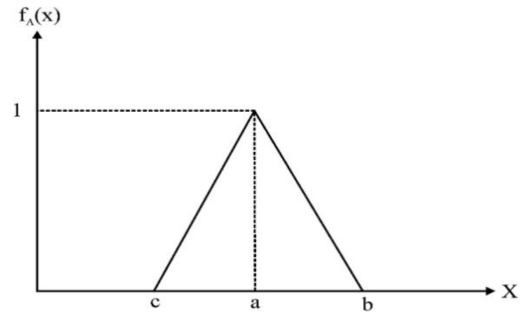

Figure 1: Membership function of a triangular fuzzy number

Table 1: Linguistic terms for the importance weight of each criterion

Linguistic term	Very low (VL)	Low (L)	Medium (M)	High (H)
Membership function	(0, 0, 0.3)	(0, 0.3, 0.5)	(0.2, 0.5, 0.8)	(0.5, 0.7, 1)

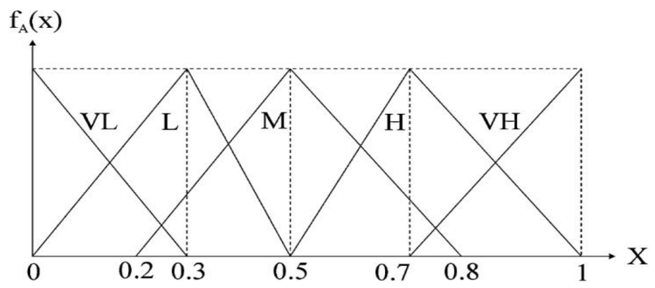

Figure 2: Membership functions for importance weight of each criterion

.

Table 2: Linguistic terms for the rating of software engineering metrics

Linguistic term	Very poor (VP)	Poor (P)	Fair (F)	Good (G)	Very Good (VG)
Membership function	(0,0,0.2)	(0,0.2,0.4)	(0.3,0.5, .7)	(0.6,0.8, 1)	(0.8, 1, 1)

Triplets as A_1 = *(c*$_1$, *a*$_1$, *b*$_1$*)* and A_2 = *(c*$_2$, *a*$_2$, *b*$_2$*)*, the addition and multiplication operations of A_1 and A_2 can be expressed as follows:

Addition: if $\oplus$ represents addition.

Equation 3-2

$$A_1 \oplus A_2 : (c_1, a_1, b_1) \oplus (c_2, a_2, b_2) = (c_1 \oplus c_2, a_1 \oplus a_2, b_1 \oplus b_2)$$

Multiplication: if $\otimes$ shows multiplication.

Equation 3-3

$$A_1 \otimes A_2 : (c_1, a_1, b_1) \otimes (c_2, a_2, b_2) = (c_1 \otimes c_2, a_1 \otimes a_2, b_1 \otimes b_2), c_1 \geq 0,\ c_2 \geq 0$$

3.3 LINGUISTIC TERMS IN TRIANGULAR FUZZY NUMBERS

Fuzzy set theory is mainly related with measuring the imprecision in human opinions and perception, where verbal terms can be in well manner shown by the estimated reasoning of fuzzy set theory. The weights of different and the rating values of software engineering metrics are considered as linguistic terms in whole this thesis. A verbal term can be defined as a variable whose values are not numbers but words or sentences in natural language. The weights can be evaluated by linguistic terms such as very low, low, medium, high, and very high. These linguistic terms can be expressed via triangular fuzzy numbers, as shown in Table I, while the membership functions of the five linguistic values are shown in Figure 2.

In order to find the appropriate of different software engineering metrics versus various ranking criteria, the rating values can be fetched by verbal terms such as very poor, poor, fair, good, and very good. These verbal terms can be represented by triangular fuzzy numbers, as shown in Table II, while the membership functions of the five verbal values are shown in Figure 3. Triangular membership functions have been used in different areas of application, as well as in this paper because of their perceptive representation and ease in estimation.

3.4 A FUZZY ALGORITHM FOR SOFTWARE ENGINEERING METRICS RANKING PROBLEM

A systematic way to the software engineering metrics ranking problem, based on fuzzy set theory and multi-criteria decision analysis, is given in this part. Many ways have been projected to get together the Thoughts of experts such as mean, median, max, min, and mixed operators (Buckley, The multiple judge multiple ranking problem: A fuzzy set approach, 1984).

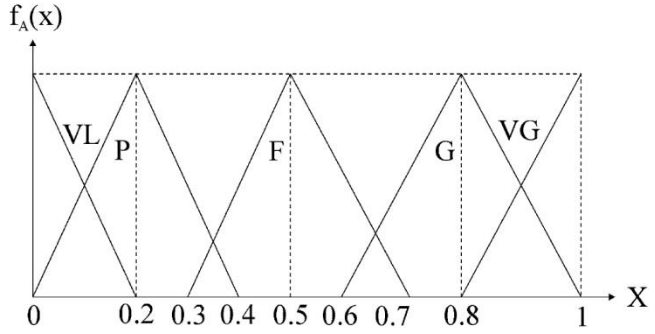

Figure 3 Membership functions for rating of software engineering metrics

Since the mean activity is the most generally used aggregation method (Chang & Chen, 1994), (Wang & Chang, 1995), (Cochran & Chen, 2005) in this study, the mean operator was used to aggregate the evaluations of experts. For software engineering metrics ranking problem, there are a group of *n* experts (*E*1, *E*2, ..., *En*), who calculate the weights of *k* criteria *(*$C_1, C_2, \dots, C_k$*)* and the grading of *m* software engineering Metrics *(* $A_1, A_2, \dots, A_m$ *)*, in each of these *k* criteria. Let W_{te} *(t* = 1, 2, . . ., *k*; *e* = 1, 2, . . ., *n)* be the weight given to C_t by expert E_e. Let R_{ite} (*i* = 1, 2, . . ., *m*; *t* = 1, 2, . . ., *k*; *e* = 1, 2, . . ., *n*). be the rating given to software engineering metric A_i by expert E_e for criterion C_t . W_t and R_{it} are defined as follows:

Equation 3-4

$$W_t = \left(\frac{1}{n}\right) \otimes (W_{t1} \oplus W_{t2} \oplus \cdots \oplus W_{t3}) = \left(\frac{1}{n}\right) \sum_{e=1}^{n} W_{te}$$

Equation 3-5

$$R_{it} = \left(\frac{1}{n}\right) \otimes (R_{it1} \oplus R_{it2} \oplus \cdots \oplus R_{itn}) = \left(\frac{1}{n}\right) = \sum_{e=1}^{n} R_{ite}$$

Where W_t is the mean weight of criterion C_t and R_{it} is the total grading of software engineering metric A_i for criterion C_t .

3.5 CONVERSION OF FUZZY NUMBERS TO CRISP SCORES

Since the total evaluations are shown as triangular fuzzy numbers, a technique of changing of these fuzzy triangular numbers to crisp values is needed. There are several techniques of changing fuzzy numbers (Liou & Wang, 1992)(Kim & Park, 1990). In this Thesis, the maximizing set and minimizing set methods are used because of the simple in of use and application in previous studies (Wang & Chang, 1995),(Yeh, Deng, & Chang, 2000),(Karsak & Tolga, 2001),(Cochran & Chen, 2005).

Let F_i *(i* = 1, 2, . . ., *m)* be the fuzzy grading of *m* software engineering metrics. Chen (Chen, 1985) gave the maximizing set $M = \{(x, f_M(x)) | x \in \mathrm{R}\}$ with

Equation 3-6

$$f_M(x) = \begin{cases} \dfrac{x - x_{min}}{x_{max} - x_{min}} & x_{min} \le x \le x_{max} \\ 0 & otherwise \end{cases}$$

and, minimizing set $G=\{(x, f_G(x))|x \in \mathrm{R}\}$ with

Equation 3-7

$$f_M(x) f_G(x) = \begin{cases} \dfrac{x - x_{min}}{x_{max} - x_{min}} & x_{min} \leq x \leq x_{max} \\ 0 & otherwise \end{cases}$$

Where $x_{\min} = \inf S$, $x_{\max} = \sup S$, $S = \cup^{m}_{i=1} F_i$, $F_i = \{x| f_{Fi}(x) > 0\}$, $i = 1, 2, . . ., m$.

Further, the right utility value $U_M (F_i)$ and the left utility value $U_G (F_i)$ for software engineering metric i are given as:

Equation 3-8

$$U_M(F_i) = \sup(f_{F_i}(x) \cap f_M(x) \quad i = 1,2,3 \ldots\ldots m$$

Equation 3-9

$$U_G(F_i) = \sup(f_{F_i}(x) \cap f_G(x) \quad i = 1,2,3 \ldots\ldots m$$

And, the total utility value $U_T (F_i)$ for another i are described as:

Equation 3-10

$$U_T(F_i) = \frac{\{U_M(F_i) + 1 - U_G(F_i)\}}{2}$$

3.6 MATRIX METHOD

Each software engineering metric at this phase is characterized by multiple criteria, which require to be changed into a single number index. This single number index will be utilized to rank the software engineering metrics in order of their effect on software reliability. This value for each software engineering metric is found using matrices. The matrices provide themselves easily to handmade manipulations and are appropriate for computer processing. The deterministic values (crisp scores) of the total evaluations i.e. grading of the software engineering metrics and the associative total weights of all recognized ranking criteria are kept in a matrix that is known as ‘Criteria Matrix’. The size of this matrix is be $n \times n$ related to n criteria. The diagonal elements (a_{ii} ’s or a_i ’s) and the their elements ($a_{i\,j}$ ’s) of this matrix give the total grading of different software engineering metrics versus different ranking criteria and the comparative total weights of various ranking criteria, respectively. Hence, the criteria matrix is a made of two matrices namely ‘Software Engineering Metric Rating Matrix’ and ‘Criteria Weight Matrix’.

Software Engineering Metric Rating Matrix: This matrix is made on the basis of deterministic values (crisp values) of the total grading of the software engineering metrics versus different ranking criteria. This is a diagonal matrix whose values (a_{ii} ’s or a_i ’s) represent the aggregated ratings of different software engineering metrics versus different ranking criteria.

$$\begin{bmatrix} a_{11} & 0 & \cdots & 0 & 0 \\ 0 & a_{22} & \dots. & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots. & a_{n-1n-1} & 0 \\ 0 & 0 & \cdots & 0 & a_{nn} \end{bmatrix}$$

Criteria Weight Matrix: The Criteria Weight Matrix is formed on the basis of the aggregated weights of different criteria. The off-diagonal elements of this matrix represent the aggregated weights of the criteria e.g. the element *($a_{i\,j}$)* of this matrix will give the relative importance weight of *j* th criteria with respect to *i*th criteria. All diagonal elements of this matrix are zero because there is no significance of comparing a criterion with respect to itself. Mathematically, $a_{i\,j}$ = weight of j^{th} criteria/weight of i^{th} criteria

$$\begin{bmatrix} 0 & a_{12} & \cdots & a_{1n} \\ a_{21} & 0 & \dots. & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & 0 \end{bmatrix}$$

Thus, the ‘Criteria Matrix’ corresponding to ‘*n*’ criteria, in general, is written as:

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \dots. & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}$$

3.7 PERMANENT FUNCTION REPRESENTATION

Variable Permanent Function or simply known as Permanent is a standard matrix function that is used in combinatorial mathematics (Marcus & Minc, 1965). It is a powerful tool for multi-criteria based evaluation and ranking of the systems in ascending or descending order. The Permanent is similar to the determinant of a matrix with a difference that no negative term appears in the Permanent. Computer software is developed to determine the value of the Permanent of the 'Criteria Matrix'. The algorithm is (Garg, et al., 2011):

```
(A) $P \leftarrow 0$;    $X_i \leftarrow a_{in} - \frac{1}{2}\sum_{i,j=n}^{n} a_{ij}$;   $sgn \leftarrow -1$
(B) $sgn \leftarrow -sgn$;   $P \leftarrow sgn$,
Get next subset of $(1, 2, \ldots, n-1)$ from NEXSUB;
if empty, go to (C) and if j was deleted, then : $z \leftarrow -1$; otherwise, $z \leftarrow 1$;
$x_i \leftarrow x_i + z\ a_{ij}\ (i = 1, 2, \ldots, n)$
(C) $P \leftarrow P.x_i\ (i = 1, 2, \ldots, n)$; $p \leftarrow p + p$ if more subsets remain, to (B);
Permanent $\leftarrow 2(-1)^{n-1} p$;   EXIT.
```

```
ALGORITHM NEXSUB
( A) [First entry] $m \leftarrow 1$; $j \leftarrow 1$; $z \leftarrow 1$; exit.
 (B) [Later entry] $m \leftarrow m + 1$; $x \leftarrow m$; $j \leftarrow 0$;
(C) $j \leftarrow j+1$; $x \leftarrow x/2$;if x is an integer ,to (C).
(D) $z \leftarrow (-1)^{(x+1)/2}$ ; If==$2^n$ final exit; EXIT
```

Chapter Four: **PROCEDURE FOR RANKING OF SOFTWARE ENGINEERING METRICS**

4.1 IDENTIFICATION OF SOFTWARE ENGINEERING METRICS

The software development process grouped of five stages: analysis, design, coding, testing, and operation. In each stages there are different factors that distinguish the software development work and guide to different quality standards of the final software product. Lawrence Livermore National Laboratory (LLNL) has recognized 78 software metrics associated either directly or indirectly to software reliability. This set of 78 software metrics was decreased to 30 based on semantic consideration using structural considerations as well as other vital considerations. The present study work is based on such already recognized software engineering metrics that affects software reliability.

4.2 EXPERT IDENTIFICATION AND SELECTION

In available software engineering literature, data that could comprise the basis for ranking the set of pre-selected metrics are impossible, because of scarcity of understanding in this field. Similarly, data mining of software engineering databases has made confirmed that it is impossible in practice (Mendonsa & Basili, 2000). Due to this, faith on expert thoughts was the best way to the problem of gathering ranking data. Hence, the first action is to make a team of experts who show a extensive diversity of experiences as is got in universities/consulting firms, laboratories, or government agencies demonstrated by publications, hands on practice and running research in the field linked to the issues within study and should also be adaptable sufficient to address these issues. Some of the experts have been both in academia and in industry. Persons of the industry may have better knowledge of issues of cost and advantages, while academician may have better knowledge of dealings in investigational development and it is at the edge of technological advances. For this research, we deliberately select five experts. All experts have more than 20 years experience in the area of software reliability testing and assessment, software reliability engineering, etc. Out of these five experts, three are from software industries, one from academics, and one from the software research laboratory.

4.3 SELECTION OF RANKING CRITERIA

Software engineering metrics can be compared by means of many attributes, jointly termed ranking criteria. Examples of such attributes are: repeatability (the fact that the repeated application of a measure provides identical results), cost, credibility (the fact that a measure supports the specified goals), etc. Some efforts have been made to identify attributes of software engineering metrics with the purpose of improving the software measurement. For instance, IEEE standard 982.2 (IEEE Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software, 1988) identified additional ranking criteria such as the benefits and experience characteristics of each software engineering metric. These criteria reflect industrial considerations. The study of Lawrence *et al.* (Lawrence, Persons, Sicherman, & Johnson, 1998) based its ranking of measures on a total of eight criteria, which are cost, benefit, credibility, directness, timeliness, repeatability, experience, and validation.

Each of the ranking criteria relates to some particular aspect of the measure considered important to the objectives of the study. Using the experience gained from the literature, for the problem of identifying a single measure that can be used (per life-cycle phase) to characterize reliability, the ranking criteria need to cover the following aspects: (1) The measurement's cost effectiveness (cost and benefit). This will determine whether or not the measure will be used in a 'real' software development process. (2) The measurement's quality (whether it is reliable, repeatable, formally validated, and widely used in the industry). This will determine whether the measurement is credible. (3) The measure's relevance to reliability (the direct objective of the study). A detailed definition of ranking criteria is given in Table III. Thus, it requires developing a set of criteria and corresponding levels for the ranking of software engineering metrics.

4.4 EVALUATE SOFTWARE ENGINEERING METRICS BY THE FUZZY-BASED MATRIX METHOD

In this projected technique, the weight of each criterion and the grading of every software engineering metric are described using verbal terms, which can also be represented as triangular fuzzy numbers. The fuzzy algorithm total the experts' mind set rating for criteria, and the evaluation ratings of software engineering metrics against the ranking criteria, to fixed the 'Criteria Matrix' and to compute the value of the Permanent. The fuzzy-based matrix method consists of five phases. In the first phase, the experts want to choose the suitable verbal values and membership functions for measuring the weights of each criterion and the grading values of software engineering metrics. In the second phase, the experts evaluate

Table 3: Ranking criteria definitions.

Ranking criteria	Definition
Cost	This Criteria pay attention on the efforts needed to apply and use the measure. A model of developer was defined to show the discriminations among real development organizations. The Qualification of this ranking criterion is based on this organization's typical one-year production. This ranking criterion is qualified by the comparative needed do the measurement for the one-year development given above
Benefit	Benefits are defined to be the escaping of overheads that would be acquire if the measures are not used. It is measured by the employee that would be saved for one-year software development if the measurement is carried out
Credibility	The records provided for each measure argues that it measures

	some thought of software development or software. A measure is supposed to be credible if we think it probably to support the specified purpose. This qualification is measured by the directness of the measurements. For example, the measure estimate the recorded purpose directly, or adjoin other values and algorithms to estimate the recorded aims.
Experience	This ranking criterion shows the grade to which this quantity has been used in the industry. Then, standard of this ranking criterion is a function of the number of business uses
Repeatability	A measure is supposed to be repeatable if the repeated application of the measure by the same or various people consensus similar results. This criterion is measured by how much subjective decision is needed to do the Measurement
Validation	This ranking criterion shows the grade to which the measure has been validated by the software engineering society. The standard based on whether the measure is formally validated or not and by whom
Relevance to reliability	This ranking criterion recognizes association measure for forecasting/estimating software reliability. The level is a function of the number of software reliability forecasting or estimation techniques or models that include the measure

the weight of each ranking criteria and the grading of the software engineering metrics against criteria. In the third phase, total weights and grading of software engineering metrics are computed. In the fourth phase, the average values (fuzzy numbers) obtained from the third phase are changed into crisp scores using maximizing and minimizing set methods. In the fifth phase, the ‘Criteria Matrices’ are made for every software engineering metric and the value of the Permanent of every such ‘Criteria Matrix’ is determined. In last, the software reliability metrics are ranked in according to the values of the Permanents. The software engineering metric with the highest value of the Permanent is ranked at number 1, the next as number 2, and so on.

A user friendly computer software has been generated, which has in-built scales of linguistic terms and respective membership functions for assigning significant weights to ranking criterion and ratings of software engineering metrics by the experts. Each expert has only to give weights and ratings to each ranking criterion and software engineering metric, respectively, choosing appropriate verbal term already available in tabular form in the software and all other mathematical operations like approximating average weights and grading, conversion of verbal terms into crisp scores, making of criteria matrix and determination of Permanent, etc. as described in phases 3–5 in the above paragraph are performed automatically by the software.

Chapter Five: **A CASE STUDY**

The fuzzy-based matrix method, described in the last chapter, is illustrated with an example. Computer software has been developed for best choosing of software reliability growth models using Distance-Based Approximation method (Optimal selection and accuracy estimation of software reliability models, 2011). The selection is based on 12 software reliability model selection criteria. Optimal software reliability growth model selection (OSRGMS) is the application chosen. In this software, user starts the application by double clicking application icon in Microsoft Windows environment. He chose one or more SRGM's out of 16 available software reliability growth models, as candidate models for reliability forecasting, after providing failure data in needed format. He chooses one or more model selection criteria out of 12 pre-specified selection criteria. The user has been given with a facility to choose the parameter estimation method and optimization method. The application shows the quantitative values of the criteria's and selection criteria of chosen software reliability growth models. Further, the application shows the last results in forms of ranking of different software reliability growth models in ascending/descending way along with the intermediate results against each step of the methodology.

OSRGMS was a smart candidate for experimental system because it is user-friendly GUI-based application, and is simply used by experts from various field e.g. expert from industry, academic, etc. and it does not need any wide technical knowledge in the field for its use.

This computer software is used to rank six most widely used software engineering metrics:

(1) Cyclomatic Complexity (CC); (2) Fault Density (FD); (3) Mean Time to Failure (MTTF); (4) System Design Complexity (SDC); (5) Requirements Compliance (RC); and (6) Cohesion (CH) based on seven ranking criteria as described in Table III on suggestions of five experts.

Asses the software engineering metrics by the fuzzy-based matrix method: The weights given to the seven ranking criteria and the ratings of the six software engineering metrics against each ranking parameter by five experts assigned in linguistic terms using the weight set *W* and rating set *R*, described in Section 3.3 i.e. *W* = {Very low, Low, Medium, High, and Very high}, *R* = {Very poor, Poor, Fair, Good, and Very good} and their respective membership functions are given in Tables IV and V respectively.

Through triangular fuzzy number aggregation by Equation (4), the total weights (W_t) of the seven criteria determined by the five experts and by using Equation (5), the aggregated rating (R_{it}) of software engineering metrics A_i under each criterion C_t were found and are represented in Tables VI and VII respectively. For example, the aggregated weight of criterion C_1 (Cost) was obtained as follows:

$W_1 = 1/5[(0.2, 0.5, 0.8) \oplus (0.5, 0.7, 1) \oplus (0, 0.3, 0.5) \oplus (0.2, 0.5, 0.8) \oplus (0.5, 0.7, 1)]$

$= 1/5\ (1.4, 2.7, 4.1) = (0.28, 0.54, 0.82)$

In addition, the aggregated rating (R_{11}) of software engineering metric (A_1) under criterion (C_1) can be found as follows:

$R_{11} = 1/5[(0.6, 0.8, 1) \oplus (0.8, 1, 1) \oplus (0.6, 0.8, 1) \oplus (0.8, 1, 1) \oplus (0.6, 0.8, 1)]$

$= 1/5(3.4, 4.4, 5)$

$= (0.68, 0.88, 1)$

The crisp scores of these aggregated values (fuzzy numbers) are obtained using conversion method as described in Section 3.5 i.e. Equations (6)–(10) and are shown in Table VIII. The ‘Criteria Matrices’ are formed for each of the software engineering metric and the value of the Permanent

Table 4 Linguistic assessments and membership functions for ranking criteria.

	$E1$	$E2$	$E3$	$E4$	$E5$
Cost	M (0.2,0.5,0.8)	H (0.5,0.7,1)	L (0,0.3,0.5)	M (0.2,0.5,0.8)	H (0.5,0.7,1)
Benefit	H (0.5,0.7,1)	VH (0.7,1,1)	M (0.2,0.5,0.8)	H (0.5,0.7,1)	VH (0.7,1,1)
Repeatability	L (0,0.3,0.5)	M (0.2,0.5,0.8)	H (0.5,0.7,1)	M (0.2,0.5,0.8)	M (0.2,0.5,0.8)
Creditability	VL (0,0,0.3)	L (0,0.3,0.5)	M (0.2,0.5,0.8)	L (0,0.3,0.5)	VL (0,0,0.3)
Validation	L (0,0.3,0.5)	L (0,0.3,0.5)	M (0.2,0.5,0.8)	L (0,0.3,0.5)	M (0.2,0.5,0.8)
Experience	M (0.2,0.5,0.8)	M (0.2,0.5,0.8)	M (0.2,0.5,0.8)	L (0,0.3,0.5)	VL (0,0,0.3)
Relevance to Reliability	M (0.2,0.5,0.8)	H (0.5,0.7,1)	L (0,0.3,0.5)	M (0.2,0.5,0.8)	H (0.5,0.7,1)

Table 5: Linguistic assessments and membership functions for software engineering metrics

		Cost	Benefit	Repeatability	Creditability	Validation	Experience	Relevance to reliability
CC	*E*1	G(0.6,0.8,1)	F(0.3,0.5,0.7)	F(0.3,0.5,0.7)	VG(0.8,1,1)	F(0.3,0.5,0.7)	G(0.6,0.8,1)	G(0.6,0.8,1)
	*E*2	VG(0.8,1,1)	F(0.3,0.5,0.7)	P(0,0.2,0.4)	VG(0.8,1,1)	G(0.6,0.8,1)	VG(0.8,1,1)	G(0.6,0.8,1)
	*E*3	G(0.6,0.8,1)	P(0,0.2,0.4)	VP(0,0,0.2)	G(0.6,0.8,1)	F(0.3,0.5,0.7)	G(0.6,0.8,1)	P(0,0.2,0.4)
	*E*4	VG(0.8,1,1)	G(0.6,0.8,1)	P(0,0.2,0.4)	VG(0.8,1,1)	G(0.6,0.8,1)	F(0.3,0.5,0.7)	VG(0.8,1,1)
	*E*5	G(0.6,0.8,1)	F(0.3,0.5,0.7)	G(0.6,0.8,1)	G(0.6,0.8,1)	F(0.3,0.5,0.7)	G(0.6,0.8,1)	G(0.6,0.8,1)
FD	*E*1	VG(0.8,1,1)	F(0.3,0.5,0.7)	F(0.3,0.5,0.7)	G(0.6,0.8,1)	G(0.6,0.8,1)	G(0.6,0.8,1)	G(0.6,0.8,1)
	*E*2	G(0.6,0.8,1)	G(0.6,0.8,1)	G(0.6,0.8,1)	F(0.3,0.5,0.7)	F(0.3,0.5,0.7)	VG(0.8,1,1)	G(0.6,0.8,1)
	*E*3	VG(0.8,1,1)	G(0.6,0.8,1)	VP(0,0,0.2)	G(0.6,0.8,1)	G(0.6,0.8,1)	G(0.6,0.8,1)	VG(0.8,1,1)
	*E*4	VG(0.8,1,1)	F(0.3,0.5,0.7)	P(0,0.2,0.4)	G(0.6,0.8,1)	G(0.6,0.8,1)	G(0.6,0.8,1)	G(0.6,0.8,1)
	*E*5	G(0.6,0.8,1)	F(0.3,0.5,0.7)	P(0,0.2,0.4)	F(0.3,0.5,0.7)	F(0.3,0.5,0.7)	G(0.6,0.8,1)	G(0.6,0.8,1)
MTTF	*E*1	G(0.6,0.8,1)	G(0.6,0.8,1)	G(0.6,0.8,1)	G(0.6,0.8,1)	G(0.6,0.8,1)	G(0.6,0.8,1)	G(0.6,0.8,1)
	*E*2	F(0.3,0.5,0.7)	VG(0.8,1,1)	G(0.6,0.8,1)	F(0.3,0.5,0.7)	F(0.3,0.5,0.7)	F(0.3,0.5,0.7)	G(0.6,0.8,1)
	*E*3	F(0.3,0.5,0.7)	G(0.6,0.8,1)	G(0.6,0.8,1)	G(0.6,0.8,1)	P(0,0.2,0.4)	G(0.6,0.8,1)	G(0.6,0.8,1)
	*E*4	F(0.3,0.5,0.7)	VG(0.8,1,1)	VG(0.8,1,1)	G(0.6,0.8,1)	F(0.3,0.5,0.7)	G(0.6,0.8,1)	VG(0.8,1,1)
	*E*5	P(0,0.2,0.4)	G(0.6,0.8,1)	G(0.6,0.8,1)	F(0.3,0.5,0.7)	F(0.3,0.5,0.7)	F(0.3,0.5,0.7)	G(0.6,0.8,1)
SDC	*E*1	F(0.3,0.5,0.7)	F(0.3,0.5,0.7)	P(0,0.2,0.4)	G(0.6,0.8,1)	P(0,0.2,0.4)	G(0.6,0.8,1)	F(0.3,0.5,0.7)
	*E*2	F(0.3,0.5,0.7)	F(0.3,0.5,0.7)	F(0.3,0.5,0.7)	G(0.6,0.8,1)	F(0.3,0.5,0.7)	F(0.3,0.5,0.7)	F(0.3,0.5,0.7)
	*E*3	G(0.6,0.8,1)	P(0,0.2,0.4)	F(0.3,0.5,0.7)	F(0.3,0.5,0.7)	F(0.3,0.5,0.7)	F(0.3,0.5,0.7)	G(0.6,0.8,1)
	*E*4	P(0,0.2,0.4)	G(0.6,0.8,1)	F(0.3,0.5,0.7)	G(0.6,0.8,1)	P(0,0.2,0.4)	P(0,0.2,0.4)	P(0,0.2,0.4)
	*E*5	F(0.3,0.5,0.7)	P(0,0.2,0.4)	VP(0,0,0.2)	G(0.6,0.8,1)	P(0,0.2,0.4)	P(0,0.2,0.4)	F(0.3,0.5,0.7)
RC	*E*1	P(0,0.2,0.4)	F(0.3,0.5,0.7)	F(0.3,0.5,0.7)	F(0.3,0.5,0.7)	G(0.6,0.8,1)	F(0.3,0.5,0.7)	F(0.3,0.5,0.7)
	*E*2	VP(0,0,0.2)	F(0.3,0.5,0.7)	F(0.3,0.5,0.7)	G(0.6,0.8,1)	F(0.3,0.5,0.7)	P(0,0.2,0.4)	F(0.3,0.5,0.7)
	*E*3	VP(0,0,0.2)	VP(0,0,0.2)	G(0.6,0.8,1)	F(0.3,0.5,0.7)	F(0.3,0.5,0.7)	F(0.3,0.5,0.7)	P(0,0.2,0.4)
	*E*4	VP(0,0,0.2)	P(0,0.2,0.4)	F(0.3,0.5,0.7)	G(0.6,0.8,1)	G(0.6,0.8,1)	G(0.6,0.8,1)	F(0.3,0.5,0.7)
	*E*5	P(0,0.2,0.4)	P(0,0.2,0.4)	F(0.3,0.5,0.7)	F(0.3,0.5,0.7)	G(0.6,0.8,1)	P(0,0.2,0.4)	P(0,0.2,0.4)
CH	*E*1	F(0.3,0.5,0.7)	G(0.6,0.8,1)	G(0.6,0.8,1)	F(0.3,0.5,0.7)	F(0.3,0.5,0.7)	F(0.3,0.5,0.7)	G(0.6,0.8,1)
	*E*2	G(0.6,0.8,1)	G(0.6,0.8,1)	G(0.6,0.8,1)	P(0,0.2,0.4)	G(0.6,0.8,1)	P(0,0.2,0.4)	G(0.6,0.8,1)
	*E*3	P(0,0.2,0.4)	F(0.3,0.5,0.7)	F(0.3,0.5,0.7)	F(0.3,0.5,0.7)	F(0.3,0.5,0.7)	P(0,0.2,0.4)	F(0.3,0.5,0.7)
	*E*4	F(0.3,0.5,0.7)	F(0.3,0.5,0.7)	F(0.3,0.5,0.7)	F(0.3,0.5,0.7)	F(0.3,0.5,0.7)	G(0.6,0.8,1)	F(0.3,0.5,0.7)
	*E*5	F(0.3,0.5,0.7)	VG(0.8,1,1)	VG(0.8,1,1)	P(0,0.2,0.4)	G(0.6,0.8,1)	F(0.3,0.5,0.7)	G(0.6,0.8,1)

Table 6: Aggregated weights *(W_t)* of ranking criteria

Criteria	Cost	Benefit	Repeatability	Creditability	Validation	Experience	Relevance to Reliability
w_t	0.28,0.54,0.82	0.52,0.78,0.96	0.22,0.5,0.78	0.04,0.22,0.48	0.08,0.38,0.62	0.12,0.36,0.64	0.28,0.54,0.82

Table 7: Aggregated rating *(R_{it})* of software engineering metrics

	Cost	Benefit	Repeatability	Creditability	Validation	Experience	Relevance to Reliability
CC	0.68,0.88,1	0.3,0.5,0.7	0.18,0.3, 0.54	0.72,0.92,1	0.42,0.62,0.82	0.58,0.78,0.94	0.52,0.72,0.88
FD	0.72,0.92,1	0.42,0.62,0.82	0.18,0.34,0.54	0.48,0.68,0.88	0.48,0.68,0.88	0.64,0.84,1	0.64,0.84,1
MTTF	0.3,0.5,0.7	0.68,0.88,1	0.64,0.84,1	0.48,0.68,0.88	0.3,0.5,0.7	0.48,0.68,0.88	0.64,0.84,1
SDC	0.3,0.5,0.7	0.24,0.44,0.64	0.18,0.34,0.54	0.54,0.74,0.94	0.12,0.32,0.52	0.24,0.44,0.64	0.3,0.44,0.64
RC	0,0.08,0.28	0.12,0.28,0.48	0.36,0.56,0.76	0.42,0.62,0.82	0.48,0.68,0.88	0.24,0.44,0.64	0.18,0.38,0.58
CH	0.3,0.5,0.7	0.52,0.72,0.88	0.52,0.72,0.88	0.18,0.38,0.58	0.42,0.62,0.82	0.24,0.44,0.64	0.48,0.68,0.88

Table 8: Crisp scores of software engineering metrics.

	Cost	Benefit	Repeatability	Creditability	Validation	Experience	Relevance to Reliability
CC	0.813095	0.5	0.371552	0.846296	0.6	0.730172	0.7056
FD	0.846296	0.6	0.371552	0.65	0.65	0.781034	0.8188
MTTF	0.5	0.813095	0.781034	0.65	0.5	0.65	0.8188
SDC	0.5	0.45	0.371552	0.7	0.35	0.45	0.4681
RC	0.153704	0.32069	0.55	0.6	0.65	0.45	0.3800
CH	0.5	0.67931	0.67931	0.4	0.6	0.45	0.6800
Criteria	0.534598	0.716303	0.5	0.283697	0.396154	0.395161	0.5465

of each such ‘Criteria Matrix’ is known using computer software. For example, the ‘Criteria Matrix’ made for software engineering metric, CC, is given as:

$$\begin{bmatrix} 0.8478 & 1.3836 & 0.9149 & 0.4462 & 0.6596 & 0.6872 & 1 \\ 0.7227 & 0.5 & 0.6612 & 0.3225 & 0.4767 & 0.4967 & 0.7227 \\ 1.093 & 1.5123 & 0.3549 & 0.4877 & 0.721 & 0.7512 & 1.093 \\ 2.2412 & 3.1009 & 2.0505 & 0.8692 & 1.4784 & 1.5402 & 2.2412 \\ 1.516 & 2.0976 & 1.387 & 0.6764 & 0.62 & 1.0418 & 1.516 \\ 1.4551 & 2.0133 & 1.3313 & 0.6493 & 0.9598 & 0.7699 & 1.4551 \\ 1 & 1.3836 & 0.9149 & 0.4462 & 0.6596 & 0.6872 & 0.7056 \end{bmatrix}$$

Finally, the ranking values of all six software engineering metrics and their corresponding rankings so found are shown in Table IX.

Table 9: Ranking values and ranks of the software engineering metrics

Software engineering metric	Ranking values	Rank #
CC	3602.431908	3
FD	3694.611648	2
MTTF	3758.057821	1
SDC	2951.339775	5
RC	2932.999554	6
CH	3293.575558	4

.

Table 10: Comparison with other methods

	Proposed fuzzy based matrix Method		Rank based on expert opinion		Rank based on ANOVA method		Rank based on AHP method	
	Ranking		Ranking		Ranking		Ranking	
Software engineering metric	Values	Rank	Values	Rank	values	Rank	values	Rank
CC	3602.4319	3	0.7867	3	0.3208	3	0.0367	2
FD	3694.6116	2	0.8156	2	0.3393	2	0.0357	3
MTTF	3758.0578	1	0.8367	1	0.3636	1	0.0424	1
SDC	2951.3397	5	0.5478	5	0.2376	5	0.0296	5
RC	2932.9995	6	0.5244	6	0.2334	6	0.0246	6
CH	3293.5755	4	0.6744	4	0.3099	4	0.0311	4

.

Table 11: Input required in AHP

		Cost	Benefit	Repeatability	Creditability	Validation	Experience	Relevance to reliability
CC	$E1$	0.800	0.500	0.500	0.967	0.500	0.800	0.800
	$E2$	0.967	0.500	0.200	0.967	0.800	0.967	0.800
	$E3$	0.800	0.200	0.033	0.800	0.500	0.800	0.200
	$E4$	0.967	0.800	0.200	0.967	0.800	0.500	0.967
	$E5$	0.800	0.500	0.800	0.800	0.500	0.800	0.800
FD	$E1$	0.967	0.500	0.500	0.800	0.800	0.800	0.800
	$E2$	0.800	0.800	0.800	0.500	0.500	0.967	0.800
	$E3$	0.967	0.800	0.033	0.800	0.800	0.800	0.967
	$E4$	0.967	0.500	0.200	0.800	0.800	0.800	0.800
	$E5$	0.800	0.500	0.200	0.500	0.500	0.800	0.800
MTTF	$E1$	0.800	0.800	0.800	0.800	0.800	0.800	0.800
	$E2$	0.500	0.967	0.800	0.500	0.500	0.500	0.800
	$E3$	0.500	0.967	0.967	0.800	0.500	0.800	0.800
	$E4$	0.500	0.967	0.967	0.800	0.500	0.800	0.967
	$E5$	0.200	0.800	0.800	0.500	0.500	0.500	0.800
SDC	$E1$	0.500	0.500	0.200	0.800	0.200	0.800	0.500
	$E2$	0.500	0.500	0.500	0.800	0.500	0.500	0.500
	$E3$	0.800	0.200	0.500	0.500	0.500	0.500	0.800
	$E4$	0.200	0.800	0.500	0.800	0.200	0.200	0.200
	$E5$	0.500	0.200	0.033	0.800	0.200	0.200	0.500
RC	$E1$	0.200	0.500	0.500	0.500	0.800	0.500	0.500
	$E2$	0.033	0.500	0.500	0.800	0.500	0.200	0.500
	$E3$	0.033	0.033	0.800	0.500	0.500	0.500	0.200
	$E4$	0.433	0.200	0.500	0.800	0.800	0.800	0.500
	$E5$	0.200	0.200	0.500	0.500	0.800	0.200	0.200
CH	$E1$	0.500	0.800	0.800	0.500	0.500	0.500	0.800
	$E2$	0.800	0.800	0.800	0.200	0.800	0.200	0.800
	$E3$	0.200	0.500	0.500	0.500	0.500	0.200	0.500
	$E4$	0.500	0.500	0.500	0.500	0.500	0.800	0.500
	$E5$	0.500	0.967	0.967	0.200	0.800	0.500	0.800

Table 12: Input required in AHP for weights.

		Cost	Benefit	Repeatability	Creditability	Validation	Experience	Relevance to Reliability
Cost	*E*1	1	0.6818	1.875	7.5	1.875	1	1
	*E*2	1	0.7857	1.4667	2.75	2.75	1.4667	1
	*E*3	1	0.5333	0.3636	0.5333	0.5333	0.5333	1
	*E*4	1	0.6818	1	1.875	1.875	1.875	1
	*E*5	1	0.7857	1.4667	11	1.4667	11	1
Benefit	*E*1	1.4667	1	2.75	11	2.75	1.4667	1.4667
	*E*2	1.2727	1	1.8667	3.5	3.5	1.8667	1.2727
	*E*3	1.875	1	0.6818	1	1	1	1.875
	*E*4	1.4667	1	1.4667	2.75	2.75	2.75	1.4667
	*E*5	1.2727	1	1.8667	14	1.8667	14	1.2727
Repeatability	*E*1	0.5333	0.3636	1	4	1	0.5333	0.5333
	*E*2	0.6818	0.5357	1	1.875	1.875	1	0.6818
	*E*3	2.75	1.4667	1	1.4667	1.4667	1.4667	2.75
	*E*4	1	0.6818	1	1.875	1.875	1.875	1
	*E*5	0.6818	0.5357	1	7.5	1	7.5	0.6818
Creditability	*E*1	0.1333	0.0909	0.25	1	0.25	0.1333	0.1333
	*E*2	0.3636	0.2857	0.5333	1	1	0.5333	0.3636
	*E*3	1.875	1	0.6818	1	1	1	1.875
	*E*4	0.5333	0.3636	0.5333	1	1	1	0.5333
	*E*5	0.0909	0.0714	0.1333	1	0.1333	1	0.0909
Validation	*E*1	0.5333	0.3636	1	4	1	0.5333	0.5333
	*E*2	0.3636	0.2857	0.5333	1	1	0.5333	0.3636
	*E*3	1.875	1	0.6818	1	1	1	1.875
	*E*4	0.5333	0.3636	0.5333	1	1	1	0.5333
	*E*5	0.6818	0.5357	1	7.5	1	7.5	0.6818
Experience	*E*1	1	0.6818	1.875	7.5	1.875	1	1
	*E*2	0.6818	0.5357	1	1.875	1.875	1	0.6818
	*E*3	1.875	1	0.6818	1	1	1	1.875
	*E*4	0.5333	0.3636	0.5333	1	1	1	0.5333
	*E*5	0.0909	0.0714	0.1333	1	0.1333	1	0.0909
Relevance to Reliability	*E*1	1	0.6818	1.875	7.5	1.875	1	1
	*E*2	1	0.7857	1.4667	2.75	2.75	1.4667	1
	*E*3	1	0.5333	0.3636	0.5333	0.5333	0.5333	1
	*E*4	1	0.6818	1	1.875	1.875	1.875	1
	*E*5	1	0.7857	1.4667	11	1.4667	11	1

Chapter Six: **RESULT AND DISCUSSION**

The associated ranking of software engineering metrics have been given in terms of the importance of their effect on software reliability. The higher the value of the permanent shows better ranking. Table IX shows that MTTF has been ranked highest because it scored very high value for three criteria namely cost, repeatability, and relevance to reliability. This metric is followed by FD and CC being ranked at 2 and 3 and are top three software engineering metrics. It implies that these metrics are prime candidates as a root of a software reliability forecasting system. The software engineering metric RC has the lowest ranking, a result because of the fact that the metrics scores very low in the cost criterion. The quantification analysis given in Table X shows the comparison of the rankings of software engineering metrics obtained by fuzzy-based matrix method with other available methods.

It is clear that the results, found using fuzzy-based matrix methodology, are consistent with the results obtained from other statistical analysis used by other researchers. However, it is easier to obtain the results by this methodology as a very small change in the Permanent leads to a more difference in the value of ranking of a software engineering metric.

Table 13: Input required in ANOVA method.

		Cost	Benefit	Repeatability	Creditability	Validation	Experience	Relevance to Reliability
CC	$E1$	0.800	0.500	0.500	0.967	0.500	0.800	0.800
	$E2$	0.967	0.500	0.200	0.967	0.800	0.967	0.800
	$E3$	0.800	0.200	0.033	0.800	0.500	0.800	0.200
	E4	0.967	0.800	0.200	0.967	0.800	0.500	0.967
	E5	0.800	0.500	0.800	0.800	0.500	0.800	0.800
FD	$E1$	0.967	0.500	0.500	0.800	0.800	0.800	0.800
	$E2$	0.800	0.800	0.800	0.500	0.500	0.967	0.800
	$E3$	0.967	0.800	0.033	0.800	0.800	0.800	0.967
	$E4$	0.967	0.500	0.200	0.800	0.800	0.800	0.800
	E5	0.800	0.500	0.200	0.500	0.500	0.800	0.800
MTTF	$E1$	0.800	0.800	0.800	0.800	0.800	0.800	0.800
	$E2$	0.500	0.967	0.800	0.500	0.500	0.500	0.800
	$E3$	0.500	0.967	0.967	0.800	0.500	0.800	0.800
	$E4$	0.500	0.967	0.967	0.800	0.500	0.800	0.967
	$E5$	0.200	0.800	0.800	0.500	0.500	0.500	0.800
SDC	$E1$	0.500	0.500	0.200	0.800	0.200	0.800	0.500
	$E2$	0.500	0.500	0.500	0.800	0.500	0.500	0.500
	$E3$	0.800	0.200	0.500	0.500	0.500	0.500	0.800
	$E4$	0.200	0.800	0.500	0.800	0.200	0.200	0.200
	$E5$	0.500	0.200	0.033	0.800	0.200	0.200	0.500
RC	$E1$	0.200	0.500	0.500	0.500	0.800	0.500	0.500
	$E2$	0.033	0.500	0.500	0.800	0.500	0.200	0.500
	$E3$	0.033	0.033	0.800	0.500	0.500	0.500	0.200
	$E4$	0.433	0.200	0.500	0.800	0.800	0.800	0.500
	$E5$	0.200	0.200	0.500	0.500	0.800	0.200	0.200
CH	$E1$	0.500	0.800	0.800	0.500	0.500	0.500	0.800
	$E2$	0.800	0.800	0.800	0.200	0.800	0.200	0.800
	$E3$	0.200	0.500	0.500	0.500	0.500	0.200	0.500
	$E4$	0.500	0.500	0.500	0.500	0.500	0.800	0.500
	$E5$	0.500	0.967	0.967	0.200	0.800	0.500	0.800
Weights (W)		0.5465	0.7562	0.5000	0.2438	0.3605	0.3756	0.5465

6.1 Validation of the results

The relative weight and ANOVA method used by Zhang and Pham (Zhang & Pham, 2000) for the analysis of data did not consist of expert biases and also no exact expert elicitation process has been described. The methodology suggested by Li and Smidts (Li & Smidts, A ranking of software engineering measures based on expert opinion, 2003) did not estimate the bias in expert inputs and think limited number of experts.

The aggregation of supremacy degree by OWA operator with quantifier-guided function and pair wise comparison of ranking parameter or criteria as suggested by Wang and Lin (Wang & Lin, 2003) also used AHP, which consists of large amount of time for computation and is also hard to score when the number of the parameter or criteria exceeds more than seven. Criteria interdependency may suffer losses due to oversimplifying the hierarchy. Further, in this methodology fuzzy preference relation is determined by hamming distance, and consensus measures are based on the decisions agreed by most of the experts.

BBN (Fenton & Neil, 1999),(Johnson & Yu, Objective software quality assessment., 1999) needs large amount of data that stop the widespread use of such methods and badly affect the accuracy of the results.

The methodology projected in this paper has taken care of almost all shortcomings of different other methodologies. It considers expert bias with no limits on the number of exerts, and does not need excess amount of data in comparison with BBN, and imparts a better modeling of vagueness and ambiguity connected with the pair wise assessment process. Further, Complexity of AHP is higher if number of levels exceeds or overextends the hierarchy. The projected fuzzy-based matrix method considers the relative weights directly with ratings of the criteria to find out the ranking values and thus improves the accuracy of the results.

Table 14: Input required in rank based on expert opinion

	Cost	Benefit	Repeatability	Creditability	Validation	Experience	Relevance to Reliability
CC	0.800	0.500	0.500	0.967	0.500	0.800	0.800
	0.967	0.500	0.200	0.967	0.800	0.967	0.800
	0.800	0.200	0.033	0.800	0.500	0.800	0.200
	0.967	0.800	0.200	0.967	0.800	0.500	0.967
	0.800	0.500	0.800	0.800	0.500	0.800	0.800
FD	0.967	0.500	0.500	0.800	0.800	0.800	0.800
	0.800	0.800	0.800	0.500	0.500	0.967	0.800
	0.967	0.800	0.033	0.800	0.800	0.800	0.967
	0.967	0.500	0.200	0.800	0.800	0.800	0.800
	0.800	0.500	0.200	0.500	0.500	0.800	0.800
MTTF	0.800	0.800	0.800	0.800	0.800	0.800	0.800
	0.500	0.967	0.800	0.500	0.500	0.500	0.800
	0.500	0.967	0.967	0.800	0.500	0.800	0.800
	0.500	0.967	0.967	0.800	0.500	0.800	0.967
	0.200	0.800	0.800	0.500	0.500	0.500	0.800
SDC	0.500	0.500	0.200	0.800	0.200	0.800	0.500
	0.500	0.500	0.500	0.800	0.500	0.500	0.500

	0.800	0.200	0.500	0.500	0.500	0.500	0.800
	0.200	0.800	0.500	0.800	0.200	0.200	0.200
	0.500	0.200	0.033	0.800	0.200	0.200	0.500
RC	0.200	0.500	0.500	0.500	0.800	0.500	0.500
	0.033	0.500	0.500	0.800	0.500	0.200	0.500
	0.033	0.033	0.800	0.500	0.500	0.500	0.200
	0.433	0.200	0.500	0.800	0.800	0.800	0.500
	0.200	0.200	0.500	0.500	0.800	0.200	0.200
CH	0.500	0.800	0.800	0.500	0.500	0.500	0.800
	0.800	0.800	0.800	0.200	0.800	0.200	0.800
	0.200	0.500	0.500	0.500	0.500	0.200	0.500
	0.500	0.500	0.500	0.500	0.500	0.800	0.500
	0.500	0.967	0.967	0.200	0.800	0.500	0.800

Table 15: Procedural comparisons of various methods

Step		Proposed fuzzy-based matrix method	Expert opinion	ANOVA method	AHP method
1.	Construct fuzzy Matrix	Fuzzy	Non-fuzzy	Non-fuzzy	Non-fuzzy
2.	Adjust attributes Values	Fuzzy	Non-fuzzy	Non-fuzzy	Non-fuzzy
3.	weight matrix	Fuzzy aggregation	Algebraic Aggregation	Not available	*N* pair wise comparison matrix
4.	Aggregation of Experts Opinion	Fuzzy aggregation	Algebraic Aggregation	Algebraic Aggregation	Compute the priority vector for *N* pair wise comparison matrix
5.	suitability indices of metrics	Fuzzy	Algebraic	Algebraic	Compute the Comprehensive priority vector
6.	Rank of metrics	Yes	Yes	Yes	Yes
7.	Computations	N	N	N	$(2N + 4)$

Required (N = number of attribute)

Algebraic means easy addition, multiplication, and arithmetic average. All these methodologies have been compared for computer software developed for most favourable selection of software reliability development models as detailed in chapter5. In order to have experimental analysis and validation of this comparison, the inputs needed for different techniques for ranking of software engineering metrics for the above illustrated example are given in Tables XI–XIV and the procedural comparison has been given in Table XV.

Chapter Seven: **CONCLUSION**

7.1 CONCLUSION

The study was conducted to rank the software engineering metrics using the state-of-art knowledge in the field of software engineering. In meticulous, a fuzzy-based matrix method (a multiple attribute decision-making method) has been developed. It is established that once a complete set of criteria and software engineering metrics have been identified, their significant weights and ratings are assigned using verbal terms using expert elicitation, and then this method can be applied for their ranking. The results obtained by this method and their comparison in Table X validate the results presented by other methods. In general, the following conclusion can be drawn:

The interdependencies of the ranking criteria have been given due consideration in the matrix method and since Permanent of criteria matrix is used; the situation of indeterminacy does not arise.

The use of fuzzy set theory improves the decision-making procedure by considering the vagueness and ambiguity prevalent in real-world system. We also found that the use of triangular fuzzy numbers made data collection, calculation, and interpretation of results easier for experts.

The computer software that has been developed for determining the aggregated weights, ratings, and Permanent of the criteria matrix is user friendly and also does not require extensive technical knowledge of software engineering metrics and/or ranking criteria. It takes a few seconds for solving a 20× 20 matrix and thus makes the methodology easier, simpler, and effective

7.2 FUTURE SCOPE OF WORK

In this work fuzzy based methodology has been used of Ranking of software engineering metrics for measuring reliability of software. This can be extended to fuzzy set theory concept in allocating appropriate work to suitable employee on the basis of interest. This can enhance productivity and efficiency of industry. Work need to be taken to make rank of employee interest and working as software engineering metrics. Thus this concept can further be utilised for other type of industries as software industry.

.

REFERENCES

"The Nature of Mathematical Programming. (n.d.).

(2014, june 17). Retrieved from NASA: http://mars.jpl.nasa.gov/msp98/news/mco990930.html

Abdel-Rahman, E. M., Ahmad, A. R. (2012). A metaheurisic bat inspired algorithm for full body human pose estimation. *Ninth Conference on Computer and Robot Vision,* (pp. 369–375).

Abraham A., C. G. (2006). Stigmergic Optimization. *Springer* .

Albrecht, A. (1979). Measuring Application Development Productivity. *In Proc of the IBM Applications Development Symposium* , (pp. 83-92).

Anish M, Kamal P and Harish M. (2010). Software Cost Estimation using Fuzzy logic. *ACM SIGSOFT Software Engineering Notes* , 1-7.

Anna Galinina, Olga Burceva, Sergei Parshutin. (2012). The Optimization of COCOMO Model Coefficients Using Genetic Algorithms. *Information Technology and Management Science* , 45-52.

Arnold, D. N. (2014, june 16). Retrieved from The Patriot Missile Failure: https://www.ima.umn.edu/~arnold/disasters/patriot.html

Briand, L., Freimut, B., & Vollei, F. (Eds.). (2000). Assessing the cost-effectiveness of inspections by combining project data and expert opinion. *Proceedings of the 11th International Symposium on Software Reliability Engineering*, *23*, pp. 246–258. San Jose, CA, U.S.A.

Bailador, G., & Trivi, G. ˜. (2010). Pattern recognition using temporal fuzzy automata. *Fuzzy Sets and Systems* , 37–55.

Banks A., J. V. (2007). A Review of Particle Swarm Optimization- Part I: Background and Development, Natural Computation. *springer* , 467–484.

Basili, J. B. (1981). A meta model for software development resource expenditures. *Fifth International conference on software Engineering*, (pp. 107-129).

Boehm., B. (1981). *Software Engineering Economics.* New Jersey.

Bora, T. C. (2012). Bat-inspired optimization approach for the brushless DC wheel motor problem. *IEEE Trans. Magnetics* , 947-950.

Brajesh Kumar Singh, S. T. (2013). Tuning of Cost Drivers by Significance Occurrences and Their Calibration with Novel Software Effort Estimation Method. *Advances in Software Engineering* .

Buckley, J. (1984). The multiple judge multiple ranking problem: A fuzzy set approach. *Fuzzy Sets and Systems* , 25–37.

C.F, K. (1996). An Empirical Validation of Software Cost Estimation Models. *ACM* , 416-429.

Chang, P., & Chen, Y. (1994). A fuzzy multi-criteria decision making method for technology transfer strategy selection in biotechnology. *Fuzzy Sets and Systems* , 131–139.

Chen, S. (1985). Ranking fuzzy numbers with maximizing set and minimizing set. *Fuzzy Sets and Systems* .

Cochran, J., & Chen, H. (2005). Fuzzy multi-criteria selection of object-oriented simulation software for production system analysis. *Computers and Operations Research* , 153–168.
DewSoft. (2014, June 17). Retrieved from SOFTWARE RELIABILITY: http://education.dewsoftoverseas.com/QE/QUickReference/Software%20Enginering/7.1.asp
DewSoft. (2014, June 17). Retrieved from SOFTWARE RELIABILITY: http://education.dewsoftoverseas.com/QE/QUickReference/Software%20Enginering/7.6.asp
Dolado, J. J. (2009). *On the Problem of the Software Cost Function,*. spain.
Du, Z. Y. (2012). Image matching using a bat algorithm with mutation. *Applied Mechanics and Materials* , 88-93.
Dubois D, Prade H. Fuzzy real algebra: Some results. (1979). *Fuzzy Sets and Systems* , 327–348.
Dyba, T. (2000). An instrument for measuring the key factors of success in software process improvement. *Empirical Software Engineering* , 357–390.
Evanco, W., & Lacovara, R. (1994). A model-based framework for the integration of software metrics. *The Journal of Systems Software* , 77–86.
F, s. (2006). Estimation of the COCOMO model parameters using genetic algorithms for NASA software projects. *Journal of computer science* , 118-123.
F. Ferrucci, C. G. (2010). Genetic programming for effort estimation: an analysis of the impact of different fitness functions. *in Proceedings of the 2nd International Symposium on Search Based Software Engineering (SSBSE '10),* (pp. 89-98). IEEE Computer Society.
Facts about COCOMO And Costar. (2012). Retrieved from http://www.softstarsystems.com/.
Fenton, N., & Neil, M. (1999). A critique of software defect prediction models. *IEEE Transactions Software Engineering* , 675–689.
Five reason why software projects fail. (2002, may 20). Retrieved from Computerworld.
Furuyama, T., Arai, Y., & Lio, K. (1997). Analysis of fault generation caused by stress during software development. . *The Journal of Systems and Software* , 13–25.
Furuyama, T., Yoshio, A., & Kazuhiko, I. (1994). Fault generation model and mental stress effect analysis. *The Journal of Systems and Software* , 31–42.
Gao, B. W. (1997). ASSESSING SOFTWARE COST ESTIMATION MODELS: CRITERIA FOR ACCURACY, CONSISTENCY AND REGRESSION. *Advanced journal of Information sciences* , 30-44.
Garg, R., Gupta, V., & Agrawal, V. (2007). Quality evaluation of thermal power plants by graph theoretical methodology. *International Journal of Power and Energy Systems* , 42–48.
Host, M., & Wohlin, C. (1998). An experimental study of individual subjective effort estimations and combinations of the estimates. *Proceeding of the 20th International Conference on Software Engineering*, (pp. 332–339). tokyo, japan.
http://en.wikipedia.org/wiki/COCOMO. (n.d.). Retrieved june 2014, from http://en.wikipedia.org/: http://en.wikipedia.org/wiki/COCOMO

(1988). *IEEE Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software*. IEEE, New York.
Jacob, L. (2014). Bat Algorithm for resource scheduling in cloud computing enviornment. *International Journal for research in applied sciences and engineering technology* .
Jamil, M. Z.-J. (2013). Improved bat algorithm for global optimization. *Applied Soft Computing* .
Johnson, G., & Yu, X. (1999). Objective software quality assessment. *Proceeding of Nuclear Science Symposium (NSS),* (pp. 1691–1698). Seattle, WA, U.S.A.
Johnson, G., & Yu, X. (1999). Objective software quality assessment. *Proceeding of Nuclear Science Symposium (NSS), Seattle, WA, U.S.A* , 1691–1698.
Karsak, E., & Tolga, E. (2001). Fuzzy multi-criteria decision-making procedure for evaluating advanced manufacturing system investments. *International Journal of Production Economics* , 49–64.
Kaufmann, A., & Gupta, M. (1988). Fuzzy Mathematical Models in Engineering and Management Science. *Elsevier Science Publisher* .
Khan, K. N. (2011). A fuzzy bat clustering method for er-gonomic screening of office workplaces,. *Advances in Intelligent and Soft Computing* , 59–66.
Khatatnech, K., & Mustafa, T. (2009). Software reliability modeling using soft computing technique. *European Journal of Scientific Research* , 154–160.
Kim, K., & Park, K. (1990). Ranking fuzzy numbers with index of optimism. *Fuzzy Sets and Systems* , 143–150.
Kitchenham, B., Linkman, S., & Law, D. (1997). DESMET: A methodology for evaluating software engineering methods and tools. *Computing and Control Engineering Journal* , 120–126.
Komarasamy, G. a. (2012). An optimized K-means clustering techniqueusing bat algorithm. *European J. Scientific Research* , 263-273.
Lau, H., Wong, C., Lau, P., Pun, K., Chin, K., & Jiang. (2003). A fuzzy multi-criteria decision support procedure for enhancing information delivery in extended enterprise networks. *Engineering Applications of Artificial Intelligence* , 1–9.
Lawrence, J., Persons, W., Sicherman, A., & Johnson, G. (1998). *Assessment of software reliability measurement methods for use in probabilistic risk assessment.* Lawrence Livermore National Laboratory, Fission Energy and Systems Safety Program. Technical Report UCRLID-136035.
Lemma, T. A., Bin Mohd Hashim, F. (2011). Use of fuzzy systems and bat algorithm for exergy modelling in a gas turbine generator,. *IEEE Colloquium* , 305–310.
Li, M., & Smidts, C. (2003). A ranking of software engineering measures based on expert opinion. *IEEE Transactions on Software Engineering* , 29(9):811–824. DOI: 10.1109/TSE.2003.1232286.
Li, M., Wei, Y., Desovski, D., Nejad, H., Ghose, S., & Cukic, B. (2004). Validation of a methodology for assessing software reliability. *Software Reliability Engineering 2004: ISSRE 2004, 15th International Symposium*, (pp. 66–76). Saint-Malo, Bretagne, France.
Liao, T. (1996). A fuzzy multi-criteria decision making method for material selection. *Journal of Manufacturing Systems* , 1–12.

Lin, J. H. (2012). A chaotic Levy flight bat algorithm for parameter estimation in nonlinear dynamic biological systems. *J.Computer and Information Technology* , 56–63.
Lin, J.-C. (2010). Applying Particle Swarm Optimization to Estimate Software Effort by Multiple Factors Software Project Clustering. *IEEE* .
Liou, T., & Wang, M. (1992). Ranking fuzzy numbers with integral value. *Fuzzy Sets and Systems* , 247–255.
M.jorgensen, K. a. (2003). A review of software surveys on software effort estimation. *International symposium on Empirical Software Engineering*, (pp. 223-230).
Marcus, M., & Minc, H. (1965). Permanents. *American Mathematics* , 571–591.
McIvor, R., McCloskey, A., Humphreys, P., & Maguire, L. (2004). Using a fuzzy approach to support financial analysis in the corporate acquisition process. *Expert Systems with Applications* , 533–547.
Mendonsa, M., & Basili, V. (2000). Validation of an approach for improving existing measurement frameworks. *IEEE Transactions on Software Engineering* , 484–499.
Michalewicz. (1992). Genetic Algorithms + Data Structures = Evolution Programs. *Springer* .
Molokken, K. F. (2007). Increasing Software Effort Estimation Accuracy- using experiance data, estimation models and checklists. *&th International conference on Quality Software*, (pp. 342-347). portland.
MR, L. (1995.). *Handbook of Software Reliability Engineering. McGraw-Hill: New York, 1995.* New York: McGraw-Hill.
Musa, J., A, I., & Okumoto, K. (1987). *Software Reliability: Measurement, Prediction, Applicationsl.* New York: McGraw-Hil.
Nakamura, R. Y. (2012). A binary bat algorithm for feature selection. *25th SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)* (pp. 291-297). IEEE Publication.
Optimal selection and accuracy estimation of software reliability models. (2011). *PhD Thesis submitted to M. D. University* . Rohtak, India.
Ordonez, J., & Haddad, H. (April 2008). The state of metrics in software industry. *Information Technology: New Generations, 2008. ITNG 2008. Fifth International Conference, DOI: 10.1109/ITNG.2008.106*, pp. 453–458. Las Vegas, NV.
P.R Srivastava, A. B. (2014). An empirical study of test effort estimation based on bat algorithm. *Int. J. Bio-Inspired Computation* , 57-70.
Parastoo, M., & Dehlen, V. (May 2009). Existing model metrics and relations to model quality. *ICSE Workshop on Software Quality.* Vancouver, BC, Canada.
Pham, H., & Zhang, X. (1999). A software cost model with warranty and risk costs. *IEEE Transactions on Computers* , 71–75.
Putnam, L. (1978). A general Empirical Solution to the Macro Software Sizing and Estimating Problem. *IEEE Transactions on Software Engineering* , (pp. 345-360).
Putnam, L., & Fitzsimmons, A. (1979). Estimating software costs. *Datamation* , 171–178.
Q. Alam, P. (n.d.). Systematic Review of Effort Estimation and cost Estimation. Roorkee: Institute of management studies.
Ramamoorthy, C., & Bastani, F. (1982). Software reliability: Status and perspectives. *IEEE Transactions Software Engineering* , 8(4):354–371.

Reddy, P. (2010). Software effort estimation using Particle Swarm Optimization with inertia weight. *International journal of software Engineering* , 12-23.
Roberts, J. T., Gibson, M., Fields, K., & Rainer, J. R. (1998). Factors that impact implementing a system development methodology. *IEEE Transactions on Software Engineering* , 640–648.
S K Sehra, Y. S. (2011). SOFT COMPUTING TECHNIQUES FOR SOFTWARE PROJECT EFFORT ESTIMATION. *"International Journal of Advanced Computer and Mathematical Sciences* , 160-167.
Schneberger, S. (1997). Distributed computing environments: Effects on software maintenance difficulty. *The Journal of Systems and Software* , 101–116.
Segundo. (2001). SEER-SEM Users Manual .
Sharma, A., Kumar, R., & Grover, P. (2008). Estimation of quality for software components: An empirical approach. *ACM SIGSOFT Software Engineering Notes* , 1–10.
Shepperd, M. J. (2007). A Systematic Review of Software Development Cost Estimation Studies. *IEEE Transactions on Software Engineering* .
Sheta, S. A. (2007). Software Effort Estimation by Tuning COOCMO Model Parameters Using Differential Evolution. *IEEE congess on evolutionary computation* , 1283-1289.
Singh, R., Singh, O., & Singh, Y. (2006). A methodology for ranking of software reliability measures. *IE (I) Journal-CP* , 14–20.
Software realiabilty. (2014, June 17). Retrieved from Dew soft: http://education.dewsoftoverseas.com/QE/QUickReference/Software%20Enginering/7.2.asp
Li, H., Lu, M., & Li, Q. (Eds.). (2006). Software reliability metrics selecting method based on analytic hierarchy process. *Quality Software, 2006. QSIC 2006.Sixth International Conference*, (pp. 337–346). Beijing.
Thakur, D. (2014, June 17). *Computer Notes*. Retrieved from ecomputernotes: http://ecomputernotes.com/software-engineering/software-metrics
(2009). *The 10 laws of chaos.* The Standish group International, Inc.
Vishali, Anshu Sharma, Suchika Malik. (2014). COCOMO model Coefficients Optimization Using GA and ACO. *International Journal of Advanced Research in Computer Science and Software Engineering* , 771-776.
Wang, J., & Lin, Y. (2003). A fuzzy multi-criteria group decision making approach to select configuration items for software development. *Fuzzy Sets and Systems* , 343–363.
Wang, M., & Chang, T. (1995). Tool steel materials selection under fuzzy environment. *Fuzzy Sets and Systems* , 263–270.
Wohlin, C., Mayrhauser, A., Host, M., & Regnell, B. (2000). Subjective evaluation as a tool for learning from software project success. *Information and Software Technology ,* 983–992.
X.S., Y. (2008). *Nature-Inspired Metaheuristic Algorithms.* UK: Luniver. .
Xie, J. Z. (2013). A novel bat algorithm based on differential operator and Levy flights trajectory. *Computational Intelligence and Neuroscience* .
Yang, X. S. (2011). Bat algorithm for multi-objective optimisation. *Int. J. Bio-Inspired Computation* , 267-274.

Yang, X. S., Karamanoglu, M., Fong, S. (2012). Bat aglorithm for topology optimization in microelectronic applications. *Conference on Future Generation Communication Technology*, (pp. 150–155).
Yang, X.-S. (2010). A New Metaheuristic Bat-Inspired Algorithm. *Nature Inspired Coop-erative Strategies for Optimization (NISCO 2010)* (pp. 65-74). Springer.
Yeh, C., Deng, F., & Chang, Y. (2000). Fuzzy multicriteria analysis for performance evaluation of bus companies. *European Journal of Operational Research* , 459–473.
Zadeh, L. (1965). Fuzzy sets. Information and Control. 8:338–353.
Zhang, X., & Pham, H. (2000). An analysis of factors affecting software reliability. *The Journal of Systems and Software* , 43–56.