SELECTION OF SOFTWARE ENGG. METRICS USING DISTANCE BASED APPROACH

Dissertation submitted in

partial fulfilment of the requirement

for the award of the degree of

Master of Technology

in

Software Engineering

by

ANKUSH JAIN

University Roll No. 2K12/SWE/07

Under the Esteemed Guidance of

Dr. KAPIL SHARMA

Associate Professor, Computer Engineering Department

2013-2014

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY,

SHABAD DAULATPUR BAWANA ROAD

DELHI – 110042, INDIA

SELECTION OF SOFTWARE ENGG. METRICS USING DISTANCE BASED APPROACH

Dissertation submitted in

partial fulfilment of the requirement

for the award of the degree of

Master of Technology

in

Software Engineering

by

ANKUSH JAIN

University Roll No. 2K12/SWE/07

Under the Esteemed Guidance of

Dr. KAPIL SHARMA

Associate Professor, Computer Engineering Department, DTU

2013-2014

DEPARTMENT OF COMPUTER SCIENECE AND ENGINEEING
DELHI TECHNOLOGICAL UNIVERSITY,
SHABAD DAULATPUR BAWANA ROAD
DELHI – 110042, INDIA

DECLARATION

I hereby declare that the thesis entitled "**Selection of Software Engineering Metrics Using Distance Based Approach"** 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 Software 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.

ANKUSH JAIN
Master of Technology
(Software Engineering)
College Roll No. 2K12/SWE/07
Department of Computer Engineering
Delhi Technological University,
Delhi.

CERTIFICATE

This is to certify that the dissertation titled “**SELECTION OF SOFTWARE ENGG. METRICS USING DISTANCE BASED APPROACH** ” is a bonafide record of work done at Delhi Technological University by **ANKUSH JAIN**, Roll No. **2K12/SWE/07** for partial fulfilment of the requirements for the degree of Master of Technology in Software Engineering. This thesis was carried out under my supervision and has not been submitted elsewhere, for the award of any other degree or diploma to the best of my knowledge and belief.

(Dr. Kapil Sharma)
Associate Professor
Department of Computer Engineering
Delhi Technological University

Dated

ACKNOWELDGEMENT

First of all, I would like to express my deep sense of respect and gratitude to my Thesis supervisor **Dr. Kapil Sharma** for providing the opportunity of carrying out this thesis and being the guiding force behind this work. I am deeply indebted to him for the support, advice and encouragement he provided without which the thesis could not have been a success.

Secondly, I am grateful to **Dr. O.P Verma,** HOD, Software Engineering Department, DTU for his immense support. I would also like to acknowledge Delhi Technological University for providing the right academic resources and environment for this work to be carried out.

Last but not the least I would like to express sincere gratitude to my parents and friends for constantly encouraging me during the completion of work.

ANKUSH JAIN
University Roll no: 2K12/SWE/07
M.Tech (Software Engineering)
Department of Software Engineering
Delhi Technological University
Delhi – 110042

ABSTRACT

It is very important to select and use appropriate software metrics in software engineering. This assertion proposes a framework for selecting software engineering metrics based on Distance Based Approach (DBA) and expert judgment. Selection criteria and the metrics for selection are identified. In each development phase, the grading of metrics according to every criterion are given by experts qualitatively, and then analysed synthetically to calculate the weights of metrics using DBA. Tools and techniques for software engineering metrics selection found in the literature cannot be used with high confidence as they use a limited number of selection criteria, we used a deterministic quantitative rank criteria based on a distance based approach (DBA) method, and then applied it for evaluation, optimal selection, and selection of software engineering metrics. DBA recognizes the need for relative importance of criteria for a given application, without which inter-criterion comparison could not be accomplished.

A preliminary application is practised, and the metrics whose weights are highly ranked are recommended and analysed. The method studied in this assertion can be used to select appropriate metrics correctly, stably and systemically. Furthermore, the final selection results are accordant with engineering experience, and using the metrics recommended will make software metrics evaluation more reliable and effective.

The metrics lend themselves to mechanical manipulations and are useful for analysing and deriving systems functions expeditiously to meet the objectives. A set of selection criteria were identified, it successfully presents the results in terms of a merit value which is used to rank the Software engineering metrics.

Keywords – Software engineering metrics, Distances Based Approach (DBA), metric selection criteria

Contents

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

LIST OF EQUATION

LIST OF ABBREVIATIONS

NHPP	Non-Homogeneous Poisson Process
SRGM	Software Reliability Growth Model
DBA	Distance Based Approach
OSRGMS	Optimal Software reliability growth model selection
FD	Fault density
CC	Cyclomatic Complexity
MTTF	Mean Time to Failure
SDC	System Design Complexity
RC	Requirements Compliance
CH	Cohesion
δ	distance metrics
r_{ij}	indicator value for alternative SRGM for attribute j
$\bar{r}_j$	average of attribute j
S_j	standard deviation of attribute j
CD	Euclidean Composite distance

Chapter One: **INTRODUCTION**

1.1 OVERVIEW

1.1.1 SOFTWARE

SOFTWARE plays a critical part in both scientific and business related enterprises, but also use in daily life where it runs devices such as cars, phones, and television sets. Although advances have been made towards the production of defect and fault free software, any software required to function reliably must still undergo extensive testing and debugging. This can be a very costly and time consuming process. The success of any software is depends on the testing process. To effectively manage their budgets, managers require accurate information about how software reliability grows as a result of this process.

Software development involves creating a software system depending on requirements of the system. Requirements may be generally complex and in this condition make software projects change continuously. Software projects are changed or modified in order to better understand of the user requirements or eliminate errors. Hence, software systems are called to be complex.

1.1.2 THE "SOFTWARE CRISIS"

It has been estimated that, by 1990, fully one half of the American work force will rely on computers and software to do its daily work. As computer hardware costs continue to decline, the demand for new applications software continues to increase at a rapid rate. The existing inventory of software continues to grow, and the effort required to maintain it continues to increase as well. At the same time, there is a significant shortage of trained software professionals. Combining these factors, one might project that at some point in the not too distant future, every American worker will have to be involved in software development and maintenance. Meanwhile, the software development scene is often characterized by:

- schedule and cost estimates that are grossly inaccurate,
- software of poor quality, and
- a productivity rate that is increasing more slowly than the demand for software.

This situation has often been referred to as the “software crisis.

1.1.3 THE NEED FOR SOFTWARE METIRCS

The software crisis must be addressed and, to the scope possible, resolved. To do so requires more correct agenda and cost estimates, better quality products, and higher productivity. All these can be achieved through more effective software management, which, in turn, can be facilitated by the improved use of software metrics.

Current software management is useless because software development is very complex, and we have few well defined, reliable measures of either the process or the product to guide and evaluate development. Thus, accurate and effective guessing, planning, and control are nearly impossible to achieve. Improvement of the management process depends upon improved ability to identify, measure, and control essential parameters of the development process.

This is the goal of software metrics the identification and measurement of the essential parameters that affect software development. Software metrics and models have been proposed and used for some time.

Metrics, however, have hardly ever been used in any regular, methodical fashion. Recent results indicate that the reliable implementation and application of a software metrics program can help achieve better management results, both in the short run (for a given project) and in the long run (improving productivity on future projects). Most software metrics cannot importantly be discussed in isolation from such metrics programs. Better use of existing metrics and development of improved metrics appear to be important factors in the resolution of the software crisis.

The IBM report only 10% project succest in their life cycle, whereas majority of them are not similar to the desired satisfacication of the customer. Developing a software that is trust worthy is precious, but what is the cost of developing software that is substandard ? A well known miss happening in 1999, NASA lost the Mars Lander due to an error made by the development team who deliberated software to estimate distances in Metric and English units but failed to design software to make proper conversions between the two. NASA lost valuable time, money and pride on a simple error that should have been detected prior to deployment of the Mars Lander.

NASA's loss was significant, but a larger misfortune occurred in 1991 when a Patriot Scud missile used during the Persian Gulf War failed to detect an incoming scud missile. The Patriot Scud, which was earlier hailed for its accuracy, had a small rounding error in the timer (approximately 0.000000095 seconds for every second of time the Patriot Missile was in use). The timer, which was necessary for calculating distances of incoming scud missiles, gathered an error of approximately 0.34 seconds over 100 hours of operation which was enough to fail to detect an incoming scud missile. The Patriot Scud failure cost the lives of 28 American soldiers.

Software life cycle is the process of developing and changing software systems. A software life cycle consists of all the events and products that are needed to develop a software system. Due to the fact that software systems are complex, life cycle models are given as Build and Fix Model, Water Fall Model, Increment Process Model, Evolutionary Process Model, Unified Process. These models contain different stage of software development life cycle. In which the development of the software tend to enable developers to cope with software complexity. Life cycle models expose the software development activities and their dependencies in order to make them more visible and manageable.

Software failures have become the most vital factor that terminates the services and affect proper functioning of the whole system. Therefore, it is very important and urgent to eliminate as many potential problems in software as possible. The software development process has become gradually time incontrollable and expensive due to the complexity of software systems. In the meantime, the need for extremely reliable software system is ever increasing. How to enhance the quality of the software systems and reduce the cost to an acceptable level becomes a major concern of today's computing industry. Methods of applying reliability and cost models to the software development practice are highly desired.

Software reliability may be defined as the probability that software will not cause a failure of a system for a specified time under specified conditions, and is one of the most important appearances of quality. Most of the software reliability models that were designed to quantify the likelihood of software failure are based on software failure observations made during test or operation (1).

The classical software reliability models may not apply in certain cases where it may be impossible to see a suitable number of failures. It may also be the case that some industries or research laboratories elect to measure other software engineering metrics like Cyclomatic complexity and fault density (FD).etc. Thus, software reliability may have to be estimated a posteriori from available sets of software engineering metrics. To report this topic, it may be considered that the product characteristics and the operational environment are two factors that underwrite to determining software reliability.

Further, the project characteristics, like functional size, the kind of application etc., and the development characteristics, such as the developer's skill, project budget, tightness of schedule, methods, tools, and languages needed for the development of the product, determine product characteristics. All the above characteristics can be measured explicitly or indirectly using software engineering metrics. Therefore, an obvious inference is that 'software engineering metrics determine software reliability' (3).

Software engineering metrics depends on the following criteria such as cost, benefit, credibility, experience, repeatability, validation reliability evaluation and assurance, are ranked in terms of their capability to predict software reliability. It is very important to selection of software engineering metrics because top ranked metrics are the possible roots of complete set of metrics to obtain trustworthy reliability predictions. The ranking is also significant for software industry for better management and quality control of software development processes and hence to enhance the software quality.

Software engineering metrics depends on such as validation evaluations are ranked in the term of their customer satisfaction. It is important to rank software engineering matrices because customer satisfaction is very important for any software.

Selection of the software engineering metrics on the basis of many criteria creates a multi criteria decision making problem. The values given to selection criteria are often qualitatively described or inexactly measured. The importance of each criterion may also vary under 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 imprecise nature of software engineering metrics and the ranking criteria, there is a need to develop a multi criteria decision making method based on distance based approach. Distance based approach (4) was developed to select the optimal software metrics.

The goal of performing this empirical research is to improve the understanding of software engineering metrics that may have influence on software reliability, cost, benefit, credibility, experience etc. analyse the significance of their effects on the software. Thus, it requires developing a Distance based Approximation to systematically rank the existing software engineering metrics with respect to their impact on the prediction of software reliability.

Parastoo and Dehlen (5) discussed the use of metrics for evaluating quality. They presented an overview of proposed metrics in literature and some examples of usage.

Ordonez and Haddad (6) examined the practices of metrics in software industry and their experiences of some related organizations. These experiences show suggestion the benefits and enhancements the quality and reliability of the software.

1.2 PROBLEM STATEMENT

Despite the fact that the Software Engineering field doesn't have a unified set of metrics that the community has approved to use, it is recommended to use them. Often during the software development process, the members of the development team do not know if what they are doing is correct and they need a guide that could help them orientate further improvement the software what is important criteria and to objectively know if the improvement is being achieved. Software metrics are tools that help to track software improvement.

Most large companies dedicated to develop software, use metrics in a constant way. Many companies have created their own standards of software measurement; so, the way that metrics are applied usually varies form one company to another one. Nevertheless, as they are used in a consistently way through dissimilar projects, the software groups get many benefits from them. What to measure in regards of software process or product depends on the nature of the project, but in all cases, the customer satisfaction is the goal and measures should be taken to achieve that goal not only at delivery, but through the entire development process.

1.3 CONTRIBUTION OF THE THESIS

There is no unique metric which works in the entire development phases. Keeping several metrics on one system helps to have a handy solution that can be used on different aspects of the software development process. The project developed in this thesis provides a solution for this need by implementing metrics that could be applied on selection of the software matric based on the Distance Based Approach. The Distance Based Approach is a Software Engineering Process that captures many of the best practices in modern software development in a form that is suitable for a wide range of projects and organizations. The metrics covered in this thesis are the following:

• For the Requirements workflow, the Specificity and Completeness of Requirements

• For the Design: Cyclomatic Complexity, Function Points, Information Flow and the Bang Metric.

• For the Implementation: the Estimation of Number of Defects, the Lines of Code (LOC) and the Halstead Metrics.

• For the Test and Deployment: the Number of Defects is again.

1.4 Organization of the Thesis

This thesis is organized into five chapters. Chapter 1 is the introductory chapter where the problem statement is described. Chapter 2 presents a classification of software metrics and the related work in software industry. Chapter 3 describes the literature review. Chapter 4 describes the methodology adopt. , Chapter 5 describe the result analysis, Chapter 6 describe the conclusion and future work, Chapter 7 describe reference

Chapter Two: **SOFTWARE ENGINEERING MATRICS**

2.1 SOFTWARE METRICS

Software metrics are an integral part of the state of the practice in software engineering. More and more customers are specifying software quality metrics reporting as part of their contractual requirements. Industry standards like ISO 9000 and industry models like the Software Engineering Institute's (SEI) Capability Maturity Model Integrated (CMMI®) include measurement.

Companies are using metrics to better understand, track, control and predict software projects, processes and products. The term software metrics means different things to different people. When we buy a book or pick up an article on software metrics, the topic can vary from project cost and effort prediction and modelling, to defect tracking and root cause analysis, to a specific test coverage metric, to computer performance modelling.

These are all examples of metrics when the word is used as a noun. I prefer the activity based view taken by Goodman. He defines software metrics as, "The continuous application of measurement based techniques to the software development process and its products to supply meaningful and timely management information, together with the use of those techniques to improve that process and its products." (7) Figure 1, illustrates an expansion of this definition to include software related services such as installation and responding to customer issues. Software metrics can provide the information needed by engineers for technical decisions as well as information required by management. If a metric is to provide useful information, everyone involved in selecting, designing, implementing, collecting, and utilizing it must understand its definition and purpose. This paper outlines twelve steps to selecting, designing and implementing software metrics in order to insure this understanding.

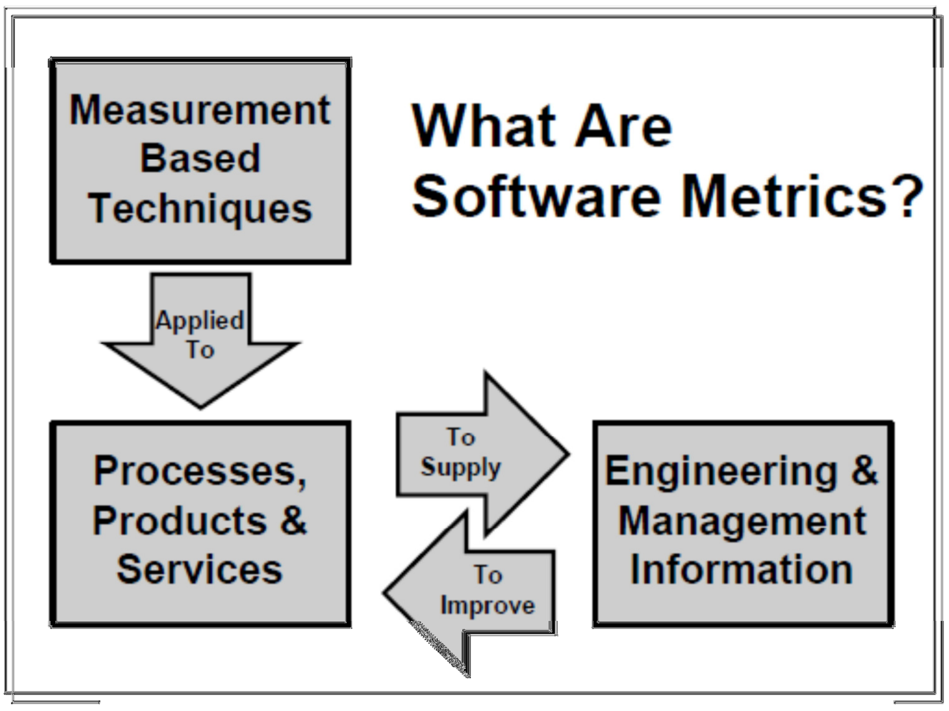

Measurement Based Techniques
What Are Software Metrics?
Applied To
Processes, Products & Services
To Supply
To Improve
Engineering & Management Information

Figure 1 Software Metrics

The use of measurement is common. We use Entity: measurements in everyday life to do such things as weigh ourselves in the morning or when we check the time of day or the distance we have travelled in our car. Measurements are used extensively in most areas of production and engineering to estimate costs, calibrate equipment, assess quality, and monitor inventories. Science and engineering disciplines depend on the rigor that measurements provide, but what does measurement really mean? According to Fenton, "measurement is the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to describe them according to clearly defined rules" (7).

Once measures are collected they are converted 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 goal of software metrics is to identify and control essential parameters that affect software development. Other objectives of using software metrics are listed below

- Measuring the size of the software quantitatively.
- Assessing the level of complexity involved.
- Assessing the strength of the module by measuring coupling.
- Assessing the testing techniques.
- Specifying when to stop testing.
- Determining the date of release of the software.
- Estimating cost of resources and project schedule.

Software metrics help project managers to gain an insight into the competence of the software process, project, and product. This is possible by collecting quality and productivity data and then analyzing and comparing these data with past averages in order to know whether quality improvements have occurred. Also, when metrics are applied in a consistent manner, it helps in project planning and project management activity. For example, schedule-based resource allocation can be effectively enhanced with the help of metrics.

An entity is a person, place, thing, event or time period. An attribute is a feature or property of the entity. To measure, we must first determine the entity. For example, we could select a car as our entity. Once we select an entity, we must select the attribute of that entity that we want to describe. For example, the car's speed or the pressure in its tires would be two attributes of a car. Finally, we must have a defined and accepted mapping system. It is meaningless to say that the car's speed is 65 or its tire pressure is 75 unless we know that we are talking about miles per hour and pounds per square inch, respectively. We will use the basic process model of input - process - output to discuss software entities. Software entities of the input type include all of the resources used for software research, development, and production.

Examples of input entities include people, materials, tools, and methods. Software entities of the process type include software-related activities and events and are usually associated with a time factor.

Examples of process entities include defined activities such as developing a software system from requirements through delivery to the customer, the inspection of a piece of code, or the first 6 months of operations after delivery. Process entities also include time periods, which do not necessarily correspond to specific activities.

An example would be the period between 1/1/93 and 2/1/93. Software entities of the output type are the products of the software process. These include all the artefacts, deliverables, and documents that are produced.

Examples of software output entities include requirements documentation, design specifications, code (source, object & executable), test documentation (plans, scripts, specifications, cases, reports), project plans, status reports, budgets, problem reports, and software metrics.

Each of these software entities has many properties or features that we might want to measure. We might want to examine a computer's price, performance, or usability. We could look at the time or effort that it took to execute a process, the number of incidents that occurred during the process, its cost, controllability, stability, or effectiveness. We might want to measure the complexity, size, modularity, testability, usability, reliability, or maintainability of a piece of source code.

One of the challenges of software metrics is that few standardized mapping systems exist. Even for seemingly simple metrics like the number of lines of code, no standard counting method has been widely accepted. Do we count physical or logical lines of code? Do we count comments or data definition statements? Do we expand macros before counting and do we count the lines in those macros more than once? Another example is engineering hours for a project besides the effort of software engineers, do we include the effort of testers, managers, secretaries, and other support personnel?

A few metrics, which do have standardized counting criteria, include McCabe's Cycloramic Complexity and the Function Point Counting Standard from the International Function Point User Group (IFPUG). However, the selection, definition, and consistent use of a mapping system within the organization for each selected metric are critical to a successful metrics program

2.2 CLASSIFICATION OF SOFTWARE METRICS

Software Metrics are standards to determine the size of an attribute of a software product and a way to evaluate it. They can also be applied to the software process. Several books present different classification of software metrics, most of them agree on the following (8):

2.2.1 SOFTWARE PRODUCT METRICS

Product metrics are measures of the software product at any stage of its development, from requirements to installed system. Product metrics may measure the complexity of the software design, the size of the final program (either source or object code), or the number of pages of documentation produced.

2.2.2 SOFTWARE PROCESS METRICS:

These metrics measure the process in regards to the time that the project will take, cost, methodology followed and how the experience of the team members can affect these values. They can be classified as empirical, statistical, theory base and composite models.

This research project focuses on presenting software selection of engineering metrics based on distance based approach as they could be applied on the different phases of the Rational Unified Process of Software Development.

The Rational Unified Process is a Software Engineering Process that captures many of the best practices in modern software development in a form that is suitable for a wide range of projects and organizations. The Rational Unified Process can be applied for small development teams as well as for large software teams. Moreover, the Rational Unified Process is a guide for how to effectively use the Unified Modeling Language 6 (UML). The UML is an industry-standard language that allows software organizations to clearly communicate requirements, architectures and designs (9).

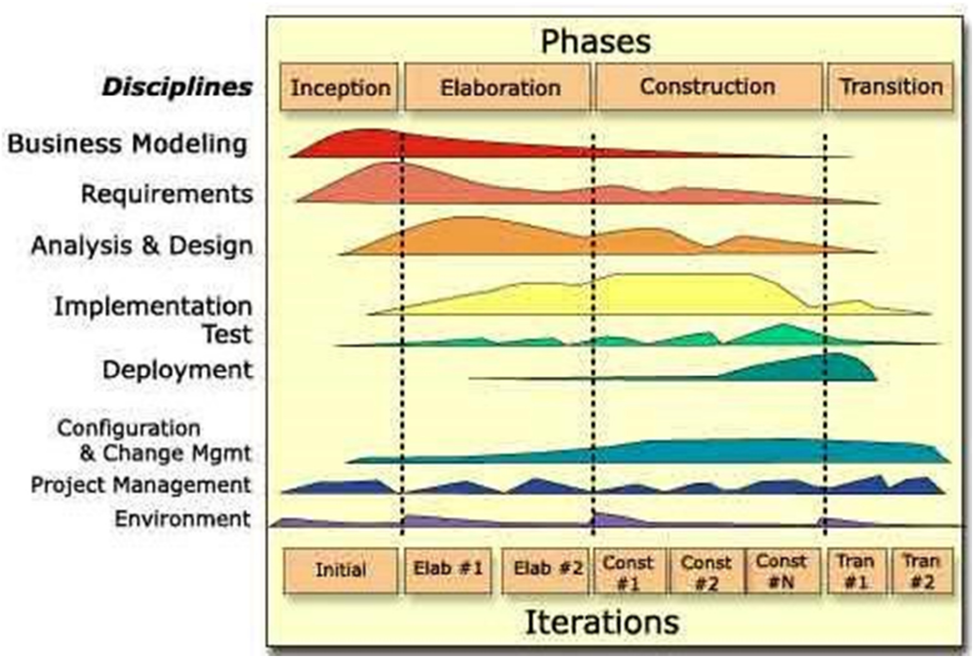


Figure 2 The Rational Unified Process,

The Rational Unified Process is shown in Figure 2.2. In the Rational Unified Process, we can distinguish two dimensions, one on the horizontal axis and another one on the vertical axis. The horizontal axis represents time and shows the dynamic aspect of the process expressed in terms of cycles, phases, iterations, and milestones. Those are the Inception phase, the Elaboration phase, the Construction phase and the Transition phase. Each phase ends with a well-defined major milestone. The vertical axis represents the static aspect of the process. A process describes who is doing what, how, and when. In the Rational Unified Process, the Workers are the ‘who’, the Activities are the ‘how’, the Artifacts are the ‘what’ and the workflows describe the ‘when’.

2.3 EXAMPLES OF SOFTWARE METRICS SYSTEMS

Many people have had the idea of concentrating software metrics that could be used during the software process, even though not all software metrics have been organized in the same manner. This section presents some examples of software metrics sets that present and implement different software metrics.

2.3.1 SOFTWARE MEASUREMENT LABORATORY

There is a very extensive and comprehensive presentation of software metrics and tools at http://irb.cs.uni-magdeburg.de/sw-eng/us/index.shtml posted by The Software Measurement Laboratory (SMLab) at the University of Magdeburg, Germany. The SMLab's team Members led by Prof. Reiner R. Dumke. The SMLab’s team has done a very good job concentrating different useful community activities related to software metrics; those go from forums, conferences and workshops, to articles and applets for

	Company/Organization	**Brief Comments**
1.	SMLab	Software Measurement Laboratory at the University of Magdeburg, Germany
2.	ZD-MIS	Zuse /Drabe Measure-Information System private company.
3.	Power Software	Private company.
4.	Charismatek Software Metrics	Private company.
5.	QSM	Quantitative Software Management is a private company.
6.	CMM	Capability Maturity Model of Software of Software Engineering Institute at Carnegie Mellon University.
7.	ISO 9000	International Organization for Standardization.
8.	Total Metrics	Consulting services.
9.	David Consulting Group	Consulting services.

Table 1 Software Metrics Related Work

metrics tools. They present a wide range of tools and topics related to software metrics and have created a large community of participants that comprises members all around the world.

2.3.2 ZD-MIS

ZD-MIS stands for Zuse / Drabe Measure-Information-System, http://home.t-online.de/home/horst.zuse/zdmis.html, which provides a 'comprehensive software test framework'. The project was initiated by Horst Zuse and Karin Drave. It comprises a large set of software metrics and a book with the fundamentals. This project is currently offered as a product that can be purchased.

2.3.3 POWER SOFTWARE

Power Software is a company (9) that provides different tools of software metrics. The tools that Power Software provides go from counting lines of code to Cyclomatic Complexity, Halstead product metrics, and Object Oriented metrics. The company also provides tools to measure the effort and project management metrics.

2.3.4 CHARISMATEK SOFTWARE METRICS

Charismatek is a company (10)that provides Metrics Software Tools and consulting services, they have developed a Function Point Tool:

'WORKBENCH^{TM,}' which has been receiving good ratings by a user satisfaction survey.

This company also has other software programs to aid in the software management process.

2.3.5 QSM

Quantitative Software Management (QSM) can be found at http://www.qsm.com/. QSM also specializes on developing software metric tools for project management and they have developed SLIM-Metrics and SLIM-Data Manager software tools that graphically allow users to see resources spent and estimation of quality for the project.

Both have a database system integrated to track changes and see the history of the project across the time.

2.3.6 OTHER QUALITY MODELS

There are other Quality Models provided by different organizations that give guidelines of software product/process improvement. One notable work is the Capability Maturity Model of Software (CMM) of the Software Engineering Institute (SEI) at Carnegie Mellon University) (11). The CMM suggests a software company evolution improvement that goes from an Initial Level of software development process, in which there is no organization; to an Optimizing Level, in which there is a continuously improving process. In the Optimizing level, the software development process and products are constantly monitored and the results are predictable.

Another important work is the International Organization for Standardization, which has a standard for quality management systems . (12)Many companies follow does standards to achieve certifications.

Other companies that provide documents and consult services for software metrics application and improvement are: Total Metrics provides consultancy services to improve software metrics practices (13)and the David Consulting Group (14) which also has a large set of articles written and a vast experience in software metrics implementation.

This chapter has presented a classification of software metrics. Software Metrics give us knowledge of the status of an attribute of the software and help us to evaluate this software attribute in an objective way. Software Metrics also helps us to latter make plans for modifications that need to be implemented in the future. In addition, to save the values obtained as history for further reference. What motivates this study of Software Metrics is to develop a software metrics framework that has a set of metrics that could be used as a stand-alone metric as needed, or as part of the pipeline of the phases through all the development process. In the next chapter, we will see how one can use software metrics in the early stages of development in order to improve the software product.

2.4 DIFFERENCE IN MEASURES, METRICS, AND INDICATORS

Metrics is often used interchangeably with measure and measurement. However, it is important to note the differences between them. Measure can be defined as quantitative indication of amount, dimension, capacity, or size of product and process attributes. Measurement can be defined as the process of determining the measure. Metrics can be defined as quantitative measures that allow software engineers to identify the efficiency and improve the quality of software process, project, and product

To understand the difference, let us consider an example. A measure is established when a number of errors is (single data point) detected in a software component. Measurement is the process of collecting one or more data points. In other words, measurement is established when many components are reviewed and tested individually to collect the measure of a number of errors in all these components. Metrics are associated with individual measure in some manner. That is, metrics are related to detection of errors found per review or the average number of errors found per unit test.

Once measures and metrics have been developed, indicators are obtained. These indicators provide a detailed insight into the software process, software project, or intermediate product. Indicators also enable software engineers or project managers to adjust software processes and improve software products, if required. For example, measurement dashboards or key indicators are used to monitor progress and initiate change. Arranged together, indicators provide snapshots of the system's performance.

2.5 MEASURED DATA

Before data is collected and used, it is necessary to know the type of data involved in the software metrics. Table lists different types of data, which are identified in metrics along with their description and the possible operations that can be performed on them

2.5.1 TYPE OF DATA MEASURED

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

Table 2 Types of data measured

2.5.1.1 NOMINAL DATA

Data in the program can be measured by placing it under a category. This category of program can be a database program, application program, or an operating system program. For such data, operation of arithmetic type and ranking of values in any order (increasing or decreasing) is not possible. The only operation that can be performed is to determine whether program 'X' is the same as program 'Y'.

2.5.1.2 ORDINAL DATA

Data can be ranked according to the data values. For example, experience in application domain can be rated as very low, low, medium, or high. Thus, experience can easily be ranked according to its rating.

2.5.1.3 INTERVAL DATA

Data values can be ranked and substantial differences between them can also be shown. For example, a program with complexity level 8 is said to be 4 units more complex than a program with complexity level 4.

2.5.1.4 RATIO DATA

Data values are associated with a ratio scale, which possesses an absolute zero and allows meaningful ratios to be calculated. For example, program lines expressed in lines of code.

It is desirable to know the measurement scale for metrics. For example, if metrics values are used to represent a model for a software process, then metrics associated with the ratio scale may be preferred.

2.6 GUIDELINES FOR SOFTWARE METRICS

Although many software metrics have been proposed over a period of time, ideal software metric is the one which is easy to understand, effective, and efficient. In order to develop ideal metrics, software metrics should be validated and characterized effectively. For this, it is important to develop metrics using some specific guidelines, which are listed below.

2.6.1 SIMPLE AND COMPUTABLE

Derivation of software metrics should be easy to learn and should involve average amount of time and effort.

2.6.2 CONSISTENT AND OBJECTIVE

Unambiguous results should be delivered by software metrics.

2.6.3 CONSISTENT IN THE USE OF UNITS AND DIMENSIONS

Mathematical computation of the metrics should involve use of dimensions and units in a consistent manner.

2.6.4 PROGRAMMING LANGUAGE INDEPENDENT

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

2.6.5 HIGH QUALITY

Effective software metrics should lead to a high-quality software product.

2.6.6 EASY TO CALIBRATE

Metrics should be easy to adapt according to project requirements.

2.6.7 EASY TO OBTAIN

Metrics should be developed at a reasonable cost.

2.6.8 VALIDATION

Metrics should be validated before being used for making any decisions.

2.6.9 ROBUST

Metrics should be relatively insensitive to small changes in process, project, or product.

2.6.10 VALUE

Value of metrics should increase or decrease with the value of the software characteristics they represent. For this, the value of metrics should be within a meaningful range. For example, metrics can be in a range of 0 to 5.

2.7 RELIABILITY METRICS

Some of the metrics, which have been used to asses software reliability are:

2.7.1 PROBABILITY OF FAILURE ON DEMAND

This is a measure of the likelihood that the system will behave in an unexpected way when some demand is made on it. It is most relevant for safety-critical systems and "nonstop" systems whose continuous operation is critical. In these systems, a measure of failure occurrence is less important that the chance that the system will not perform as expected.

2.7.2 RATE OF FAILURE OCCURRENCE (ROCOF)

This is a measure of the frequency of occurrence with which unexpected behavior is likely to be observed. For example, if the ROCOF is 2/100 this indicates that 2 failures are likely to occur in each 100 operational time units. Appropriate time units are discussed shortly. This is, possibly, the most generally useful reliability metric.

2.7.3 MEAN TIME TO FAILURE (MTTF)

This is a measure of the time between observed failures. This metric is a direct analogue of a comparable metric used in hardware reliability assessment where it reflects the lifetime of system components. In software systems, components do not wear out and, usually remain operational after a single failure. Therefore, mean time to failure is only useful in software reliability assessment when the system is stable and no changes are being made to it. In this case, it provides an indication of how long the system will remain operational before a failure occurs.

2.7.4 AVAILABILITY

This is a measure of how likely the system is to be available for use. For example, an availability of 998/1000 means that in every 1000 time units, the system is likely to be available for 998 of these. This measure is most appropriate for systems like telecommunication systems, where the repair or restart time is significant and the loss of service during time is important. (10)

No single metric is universally appropriate and the particular metric used should depend on the application domain and the expected usage of the system. For large systems, it may be appropriate to use different reliability metrics for different parts of the system. All the above characteristics can be measured explicitly or implicitly using software engineering metrics. Therefore, an obvious inference is that 'software engineering metrics determine software reliability' (3).

Parastoo and Dehlen (5) discussed the use of metrics for assessing quality. They presented an overview of proposed metrics in literature and some examples of usage. Ordonez and Haddad (6) examined the practices of metrics in software industry and experiences of some related organizations. These experiences show evidence of benefits and improvements in quality and reliability.

Software engineering metrics, used for reliability evaluation and assurance, are ranked in terms of their capability to predict software reliability. It is very important to rank software engineering metrics because top-ranked metrics are the possible roots of complete set of metrics to obtain credible reliability predictions.

The ranking is also significant for software industry for better management and quality control of software development processes and hence to enhance the software quality. Ranking of the software engineering metrics on the basis of many criteria creates a multi-criteria decision-making problem.

The values given to selection criteria are often qualitatively described or imprecisely measured. The importance of each criterion may also vary under 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 imprecise nature of software engineering metrics and the ranking criteria, there is a need to develop a multi (7)criteria decision-making method based on distance based approach.

Chapter Three: **LITERATURE REVIEW**

Roberts et al. (7) identified five factors important to implementing a system development methodology. Realizing the importance of factors that influence the software metrics, it was also pointed out that there is a need for analytical methodologies that integrate both software complexity metrics and various measures describing the software development environment (8).

Schneberger (9) presented the results of his study of the effects of distributed mputing environments on software maintenance difficulty. Furuyama et al. (11)studied factors such as working stress, development methodologies, etc. They found that different settings of these factors have statistically significant impact on the quality of final software products.

Zhang and Pham (12) conducted a survey and obtained qualitative and quantitative data from software developers and managers of 13 top companies. The relative weight and analysis of variance (ANOVA) methods were used to analyze the identified 32 factors affecting software reliability. In this study no rigorous expert elicitation process was described. The expert biases were not considered and the relative weight method was not justified.

Fenton and Neil (13) proposed a Bayesian Belief Network (BBN) model to predict software defect density, and Johnson and Yu (14) presented a BBN software quality model, based on BBN technique that determines software reliability through software engineering metrics assessment. These techniques require large amount of data and hence such techniques cannot be widely used. Moreover, accuracy of the results cannot be ensured.

Analytic Hierarchy Process (AHP) has been used to select software reliability metrics (15). It involves large amount of time for computation and is also difficult to score when the number of the criteria exceeds more than seven. Criteria interdependency may suffer losses due to oversimplifying the hierarchy and estimation of the quality for software components (16).

A limited number of expert opinion applications are found in the software engineering field. Putnam and Fitzsimmons (17) proposed a subjective estimation of the length of a program. Kitchenham et al. (18) evaluated software engineering methods and tools using subject surveys as one of the evaluation methods.

Dyba (19) utilized expert opinion to identify and rank the key factors of success in software process improvement. Wohlin et al. (20) evaluated the success of a project using subjective factors. Host and Wohlin (21) performed effort estimation by combining individual estimations performed by field experts.

Briand et al. (22) proposed an expert opinion application to the assessment of the cost effectiveness of inspection. Many researchers (15) (23) have proposed methods for ranking of software engineering measures based on expert opinion. In these studies, uncertainties and bias in the expert’s judgments are deliberately reduced. Moreover, the liner additive schemes used to aggregate the scores elicited through expert opinion are comparatively rigid, inaccurate, and also does not consider the relative weights, i.e. interdependencies of software engineering metrics.

In case of analysis through AHP it is very difficult to have pair wise comparison especially when a large number of metrics are involved and thus becomes a rather complex problem to solve. From the extensive study of the available literature, it is observed that there is a need to develop a unified method that can accommodate vagueness and ambiguity occurring during human decision making and will enable to consider all ranking criteria and their relative importance concurrently in an integrated manner for ranking of software engineering metrics. Therefore, fuzzy-based matrix method (a classical multi-attribute decision-making computation method) to deal with expert judgments qualitatively and quantitatively is proposed.

The method is more flexible, accurate, and has better sensitivity and consistency. The decision-making methods, using fuzzy set theory, have gradually gained acceptance over the last decade and their applications have also become more diverse. A complete description of these applications can be found in (24).

Goel (25), and others started describing processes for which each model would be tested to see how well the model fits the data, and predicts the future events. The assertion was that different models predict well only on certain data sets; and that by comparing the predictive quality of different models, it is possible to select the best one for a given application.

Abdel Ghaly et al. (26) Compared the predictive quality of 10 models using five different methods of comparison. They showed that different methods of model selection result in different models being chosen. Also, some of their methods were rather subjective as to which model was better than others. Clearly, a simple, objective method to select models is needed.

Khoshgoftaar (27)] suggested that the AIC could be used to select the best model. Subsequent work by Khoshgoftaar & 0Wood-cock (28) showed the application of this technique with some extensive simulation work that proved the feasibility of using the AIC for model selection.

Khos hgoftaar & Woodcock (29) proposed a method to select a reliability model among various alternatives using the log-likelihood function. They apply the method to the failure logs of a project. The method selected an S-shaped model as the most appropriate.

Chapter Four: **METHODOLOGY ADOPT**

4.1 DISTANCE BASED APPROACH

The development of the DBA method begins with defining the optimal state of the overall objective, and specifies the ide-ally good values of attributes involved in the process. The optimal state of the objective is represented by the optimum model, the OPTIMAL. The vector OP, is the set of "optimum" simultaneous attributes values. In an n-dimensional space, the vector OP is called the optimal point. For practical purposes, the optimal good value for attributes is defined as the best values which exist within the range of values of attributes.

The OPTIMAL, then, is simply the SRGM that has all the best values of attributes. It is very unlikely that a certain SRGM has the best values for all attributes. Instead, a variety of alternatives may be used to simulate the optimal state. For this reason, the OPTIMAL is not to be considered as feasible alternatives, but it is used only as ref-erence to which other alternatives are quantitatively compared. The numerical difference resulting from comparison represents the effectiveness of alternatives to achieve the optimal state of the objective function. Hence, here, the decision problem is to find a feasible solution which is as close as possible to the op-timal point. The objective function for finding such a solution can be formulated as

$$Minimize\ \delta\{Alt(x), OPTIMAL\}\ Subject\ to\ x\ \subset X \quad \text{Equation 1}$$

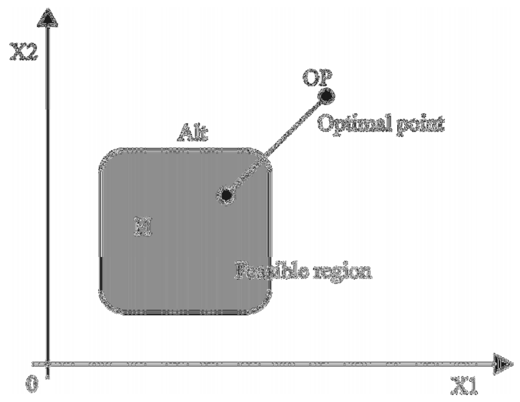


Figure 3 **distance based approach**

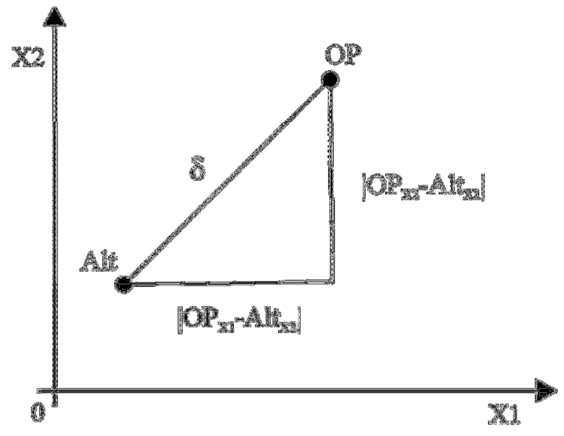


Figure 4 **distance real vector**

Where {Alt(x}}, and represent δ a Software Metrics alternative in the n-dimensional space, and the distance from the optimal point, respectively. Thus the problem, and its solutions depend on the choice of optimal point, OPTIMAL, and the distance metric, δ, used in the model. In two dimensional spaces, this solution function can be illustrated as in Fig. 1, where H is the feasible region, and the OP is the optimal point.

The DBA method determines the point in the H region which is "the closest" to the optimal point, and is graphically explained in Fig. 2 for two dimensional cases. Note that the lines $(Alt - OP)_{X1}$, and $(Alt - OP)_{X2}$ are parallel to the X1, and X2 axis respectively. Therefore, $(Alt - OP)_{X1} = |OP_{X1} - Alt_{X1}|$ and $(Alt - OP)_{X2} = |OP_{X2} - Alt_{X2}|$. Based on Pythagoras theorem, in two dimensional space, δ is

$$\delta = [(OP_{X1} - Alt_{X1})^2 + (OP_{X2} - Alt_{X2})^2]^{1/2} \qquad \text{Equation 2}$$

In general terms, the "distance δ "can be formulated as

$$\delta = \left[\sum (OP_{ij} - Alt_{ij})^2\right]^{1/2} \quad \text{Equation 3}$$

Where i=1,2,3,4,.....n= alternative SRGMs and j=1,2,3,......m =Selection attributes.

To implement the above approach, let us assume that we use have a complete set of Software Metrics consisting of 1,2,3......n SRGMs and 1,2,3,.....m selection attributes corresponding to each alternative Software Metrics. $Alt_1(r_{11}, r_{12}, r_{13}, \ldots\ldots, r_{1m})$, $Alt_2(r_{21}, r_{22}, r_{23}, \ldots\ldots, r_{2m})$, $Alt_n(r_{n1}, r_{n2}, r_{n3}, \ldots\ldots, r_{nm})$ and the OPTIMAL$(r_{b1}, r_{b2}, r_{b3}, \ldots\ldots, r_{bm})$ where r_{bm}= best value of attribute 'm'. The whole set of alternatives can be represented by the matrix

$$[r] = \begin{bmatrix} r_{11} & r_{12} & \dots & r_{1m} \\ r_{21} & r_{22} & \dots & r_{2m} \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ r_{n1} & r_{n2} & \dots & r_{nm} \\ r_{b1} & r_{b2} & \dots & r_{bm} \end{bmatrix} \qquad \text{Equation 4}$$

Thus, in this matrix, a vector in an m- dimensional space represents every Software Metrics alternatives. To case the process, and in the same time to eliminate the influence of different units of measurement, the matrix is standardized using

$$Z_{ij} = \frac{r_{ij} - \bar{r}_j}{S_j} \qquad \text{Equation 5}$$

Here $\bar{r}_{ij} = \frac{1}{n}\sum_{i=1}^{n} r_{ij}$ Equation 6

and $S_j = \left[\frac{1}{n}\sum_{i=1}^{n}(r_{ij} - \bar{r}_j)^2\right]^{1/2}$ Equation 7

Where i=1,2,3,4,…..n and j=1,2,3,……m.

$\bar{r}_j$, and S_j represent the average value and the standard deviation of each attribute for all Software Metrics, m and n represent the number of different Software Metrics attributes and number of alternate Software Metrics , respectively

$$[Z_{std}] = \begin{bmatrix} Z_{11} & Z_{12} & \dots & Z_{1m} \\ Z_{21} & Z_{22} & \dots & Z_{2m} \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ Z_{n1} & Z_{n2} & \dots & Z_{nm} \\ Z_{OP1} & Z_{OP2} & \dots & Z_{OPm} \end{bmatrix} \qquad \text{Equation 8}$$

Where$Z_{11} = r_{11} - \bar{r}_1/S_1$, $Z_{12} = r_{12} - \bar{r}_2/S_2$,$Z_{1m} = r_{1m} - \bar{r}_m/S_m$. The next step is to obtain the difference from each alternative to the reference point, the OPTIMAL, by subtracting each element of the optimal set by a corresponding element in the alternative set. This results in another interim matrix,

$$[Z_{dis}] = \begin{bmatrix} Z_{OP1} - Z_{11} & Z_{OP2} - Z_{12} & \dots & Z_{OPm} - Z_{1m} \\ Z_{OP1} - Z_{21} & Z_{OP2} - Z_{22} & \dots & Z_{OPm} - Z_{2m} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ Z_{OP1} - Z_{n1} & Z_{OP2} - Z_{n2} & \dots & Z_{OPm} - Z_{nm} \end{bmatrix} \quad \text{Equation 9}$$

Finally the Euclidean composite distance, CD between each alternative Software matrix of the optimal stat e, OPTIMAL is derived form

$$CD_{OP-Alt} = \left[\sum_{j=1}^{m} (Z_{OPj} - Z_{ij})^2\right]^{1/2} \quad \text{Equation 10}$$

Within any given set of Software Matrix's alternative this distance of each alternative to every other is obviously a composite distance. In other words, it can be referred to as the mathematical expression of several distances on each attribute in which software Matric can be compared.

4.2 SELECTION OF RANKING CRITERIA

Software engineering metrics can be compared by means of several attributes, collectively 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 (11) 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.* (12) 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 selection 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.

Ranking criteria	Definition
Cost	This Criteria concentrates on the efforts required to implement and use the measure. A model of developer was defined to reflect the differences among actual 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 staff-time required to conduct the

	measurement for the one-year production defined above
Benefit	Benefits are defined to be the avoidance of costs that would be incurred if the measures are not used. It is quantified by the staff-time that would be saved for one-year production if the measurement is carried out
Credibility	The documentation given for each measure claims that it measures some aspect of software development or software. A measure is considered to be credible if we judge it likely to support the specified goals. This criterion is quantified by the directness of the measurements. For instance, the measure evaluates the documented goal directly, or combines other quantities and algorithms to evaluate the documented goals
Experience	This ranking criterion reflects the degree to which this measure has been used in the industry. Then, level of this ranking criterion is a function of the number of commercial uses
Repeatability	A measure is considered to be repeatable if the repeated application of the measure by the same or different people results similar results. This criterion is quantified by how much subjective judgment is required to perform the Measurement.
Validation	This ranking criterion reflects the degree to which the measure has been validated by the software engineering community. The level depends on whether the measure is formally validated or not and by whom
Relevance to reliability	This ranking criterion identifies relevant measure for predicting/estimating software reliability. The level is a function of the number of software reliability prediction or

estimation methods or models that incorporate the measure

Table 3 rank criteria software metric

In this proposed method, the weight of each criterion and the rating of each software engineering metric in the form of criteria Matrix and the assessment of selection of software engineering metrics versus the ranking criteria,

4.3 SELECTION OF SOFTWARE ENGINEERING METRICS PARAMETER

The distance based approach, presented in the previous section, is illustrated with an example. Computer software has been developed for optimal selection of software reliability growth models using Distance-Based Approximation method (13). The selection is based on 12 software reliability model selection criteria. Optimal software reliability growth model selection (OSRGMS) is the application selected. In this software, user starts the application by double clicking application icon in Microsoft Windows environment. He selects one or more SRGM's out of 16 available software reliability growth models, as candidate models for reliability perdition, after supplying failure data in required format. He chooses one or more model selection criteria out of 12 predefined selection criteria. The user has been provided with a facility to select the parameter estimation technique and optimization method. The application displays the quantitative values of the parameters and selection criteria of selected software reliability growth models. Further, the application presents the final results in terms of ranking of various software reliability growth models in ascending/descending order along with the intermediate results against each step of the methodology.

OSRGMS was an attractive candidate for experimental setup because it is user-friendly GUI-based application, and is easily used by experts from different areas e.g. expert from industry, academic, etc. and it does not require any extensive 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 selection criteria is define in Table I

4.3.1 CYCLOMATIC COMPLEXITY (CC)

The Cyclomatic Complexity metric is used to objectively identify and exercise each unique node and path through the software. For unit testing, each control branching point within the source code constitutes a node: for integration testing, each unit or Computer Software Component (CSC) constitutes a node: and for CSCI requirements testing, each process bubble of a Data Flow Diagram (DFD) constitutes a node.

Complexity is used to determine the ultimate testability of the software product. For example, a procedure consisting of 25 IF statements in sequence will have over 30 million potential paths through it. The approach here is to limit the number of basis (or independent) paths that will generate all paths when taken in combination and then to test only the basis set of paths. Complexity is measured at software development outset and throughout the cycle.

Highly complex processes are identified and can be restructured to reduce the complexity so that the resulting code can be manageably tested. Testing comprises both static review and dynamic execution. The basis set of paths derived from each level of software design is used to mentally step through the design during review of software products. These paths are also used to generate test data to force execution of the paths during the corresponding testing phases.

This structured approach provides a quantification of the testing process, allowing the testing effort to be estimated, progress to be tracked and, most importantly, a definition of when testing is complete. Because complete coverage is achieved at completion, a high degree of confidence in the software is obtained.

4.3.2 FAULT DENSITY (FD)

Although seemingly straightforward, comparing the defect rates of software products involves many issues. In this metrics we try to articulate the major points. To define a rate, we first have to operationalize the numerator and the denominator, and specify the time frame. The general concept of defect rate is the number of defects over the opportunities for error (OFE) during a specific time frame. We have just discussed the definitions of software defect and failure. Because Software Quality Metrics Overview failures are defects materialized, we can use the number of unique causes of observed failures to approximate the number of defects in the software. The denominator is the size of the software, usually expressed in thousand lines of code (KLOC) or in the number of function points. In terms of time frames, various operational definitions are used for the life of product (LOP), ranging from one year to many years after the software product’s release to the general market. In our experience with operating systems, usually more than 95% of the defects are found within four years of the software’s release. For application software, most defects are normally found within two years of its release.

4.3.3 MEAN TIME TO FAILURE (MTTF)

MTTF is the mean time to the first failure under specified experimental conditions. It is calculated by dividing the total number of device • hours by the number of failures. It is important to note, at this time, that the dimensions of MTTF are not hours per failure, but rather, device • hours per failure. If each part has a 0.1% chance of failure before 1 hour then 10 parts have a 1% chance experiencing a failure by that time. The MTTF will be the same in both cases. 1 failure in 10 hours on 1 part or 1 failure in 1 hour on 10 parts both produce an MTTF of 10 device • hours. Failure rate is the conditional probability that a device will fail per unit of time. The conditional probability is the probability that a device will fail during a certain interval given that it survived at the start of the interval.(5) When failure rate is used to describe the frequency with which failures are expected to occur, the time units are typically device • hours. FITS is simply failure rate scaled from failures per device • hour to failures per billion device • hours.

4.3.4 SYSTEM DESIGN COMPLEXITY (SDC)

One of the most interesting, and most difficult, of the tasks that we may undertake in our careers as engineers or computer scientists is the design of an entire system. A system is a set of interacting parts, generally too large to be built by a single person, created for some particular purpose.

We work with systems all the time. The operating systems that control our machines are systems. The layers of hardware and software that allow the programs on these machines to interact with each other over a network are systems. Even most applications that we use are systems, whether we know it or not. As engineers, we know that the way to solve a large problem is to break it into a set of interacting smaller problems.

Each of these smaller problems can then be decomposed into even smaller problems, until after enough iteration we have a problem that can be solved on its own. Each decomposition gives us a set of components, and deciding what those components are and how they fit together is the activity of system design.

4.3.5 REQUIREMENTS COMPLIANCE (RC)

Requirements are descriptions of the services that a software system must provide and the constraints under which it must operate Requirements can range from high-level abstract statements of services or system constraints to detailed mathematical functional speculations.

Requirements engineering is the process of establishing the services that the customer requires from the system and the constraints under which it is to be developed and operated Requirements may serve a dual function:

- As the basis of a bid for a contract
- As the basis for the contract itself

Requirements Documents If a company wishes to let a contract for a large software development project it must dene its needs in a sufficiently abstract way that a solution is not predefined. The requirements must be written so that several contractors can bid for the contract, offering, perhaps, different ways of meeting the client organization's needs. Once a contract has been awarded, the contractor must write a system dentition for the client in more detail so that the client understands and can validate what the software will do. Both of these documents may be called the requirements document for the system.

4.3.6 COHESION (CH)

Most researchers and engineers agree that a good software design implies clean decomposition of the problem into modules, and the neat arrangement of these modules in a hierarchy. The primary characteristics of neat module decomposition are high cohesion and low coupling. Cohesion is a measure of functional strength of a module. A module having high cohesion and low coupling is said to be functionally independent of other modules. By the term functional independence, we mean that a cohesive module performs a single task or function.

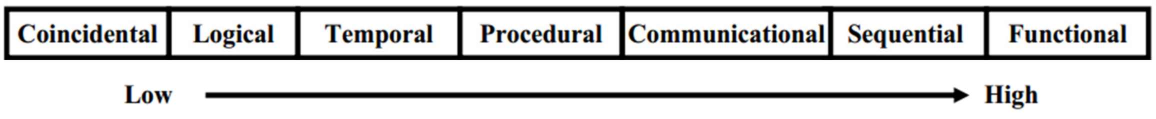


Figure 5 Types of cohesion

4.3.6.1 COINCIDENTAL COHESION

A module is said to have coincidental cohesion, if it performs a set of tasks that relate to each other very loosely, if at all. In this case, the module contains a random collection of functions. It is likely that the functions have been put in the module out of pure coincidence without any thought or design.

4.3.6.2 LOGICAL COHESION

A module is said to be logically cohesive, if all elements of the module perform similar operations, e.g. error handling, data input, data output, etc. An example of logical cohesion is the case where a set of print functions generating different output reports are arranged into a single module.

4.3.6.3 TEMPORAL COHESION

When a module contains functions that are related by the fact that all the functions must be executed in the same time span, the module is said to exhibit temporal cohesion. The set of functions responsible for initialization, start-up, shutdown of some process, etc. exhibit temporal cohesion.

4.3.6.4 PROCEDURAL COHESION

A module is said to possess procedural cohesion, if the set of functions of the module are all part of a procedure (algorithm) in which a certain sequence of steps have to be carried out for achieving an objective, e.g. the algorithm for decoding a message.

4.3.6.5 COMMUNICATIONAL COHESION

A module is said to have communicational cohesion, if all functions of the module refer to or update the same data structure, e.g. the set of functions defined on an array or a stack.

4.3.6.6 SEQUENTIAL COHESION

A module is said to possess sequential cohesion, if the elements of a module form the parts of sequence, where the output from one element of the sequence is input to the next.

4.3.6.7 FUNCTIONAL COHESION

Functional cohesion is said to exist, if different elements of a module cooperate to achieve a single function. For example, a module containing all the functions required to manage employees' pay-roll displays functional cohesion. Suppose a module displays functional cohesion, and we are asked to describe what the module does, then we would be able to describe it using a single sentence

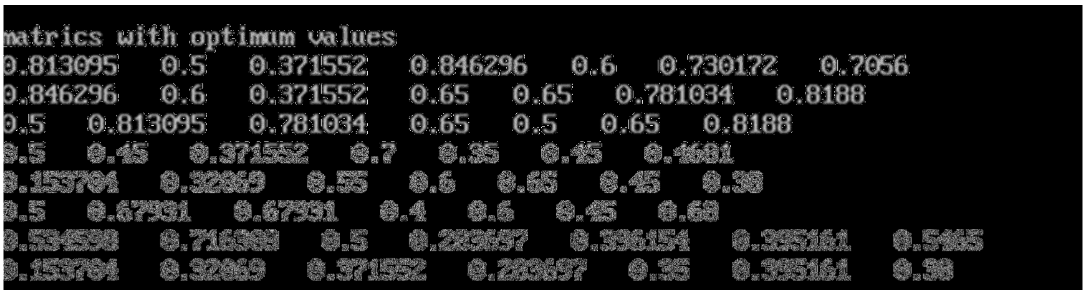
Chapter Five: **RESULT ANALYSIS**

In this section we take the crisp score matrix (33) Table VIII to evaluate the selection of optimum solution from the DBA method by the given input data set below

	Cost	Benefit	Repeatability	Creditability	Validation	Experience	Relevance to reliability
CC	0.813095	0.5	0.371552	0.846296	0.6	0.730172	0.7052
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	0.38
CH	0.5	0.67931	0.67931	0.4	0.6	0.45	0.68
Criteria	0.534598	0.716303	0.5	0.283697	0.396154	0.395161	0.5465

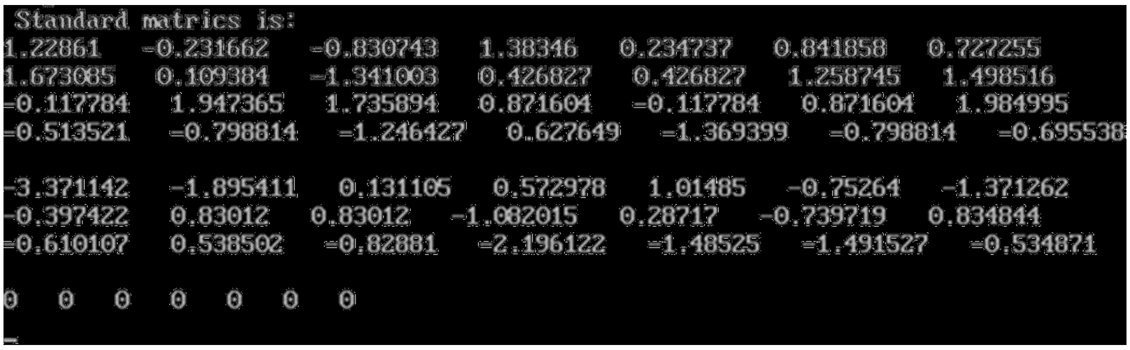
Table 4 input data set

First we select the optimal value from each column in the last row of the matix.



```
matrics with optimum values
0.813095   0.5   0.371552   0.846296   0.6   0.730172   0.7056
0.846296   0.6   0.371552   0.65   0.65   0.781034   0.8188
0.5   0.813095   0.781034   0.65   0.5   0.65   0.8188
0.5   0.45   0.371552   0.7   0.35   0.45   0.4681
0.153704   0.32069   0.55   0.6   0.65   0.45   0.38
0.5   0.67931   0.67931   0.4   0.6   0.45   0.68
0.534598   0.716303   0.5   0.283697   0.396154   0.395161   0.5465
0.153704   0.32069   0.371552   0.283697   0.35   0.395161   0.38
```

Then after we calculate the average and standard deviation then after the get the given matrix is.

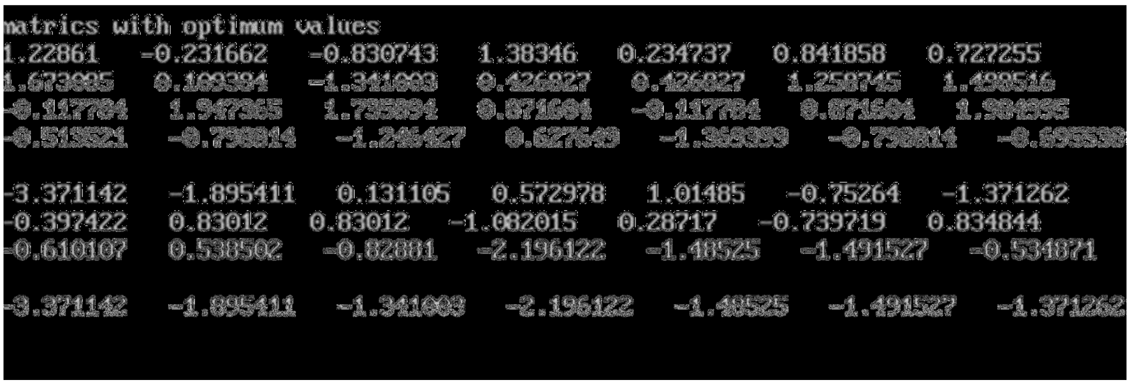


```
 Standard matrics is:
1.22861   -0.231662   -0.830743   1.38346   0.234737   0.841858   0.727255
1.673085   0.109384   -1.341003   0.426827   0.426827   1.258745   1.498516
-0.117784   1.947365   1.735894   0.871604   -0.117784   0.871604   1.984995
-0.513521   -0.798814   -1.246427   0.627649   -1.369399   -0.798814   -0.695538

-3.371142   -1.895411   0.131105   0.572978   1.01485   -0.75264   -1.371262
-0.397422   0.83012   0.83012   -1.082015   0.28717   -0.739719   0.834844
-0.610107   0.538502   -0.82881   -2.196122   -1.48525   -1.491527   -0.534871

0   0   0   0   0   0   0
_
```

Now we take the optimum value (low) in last row of this matrix.

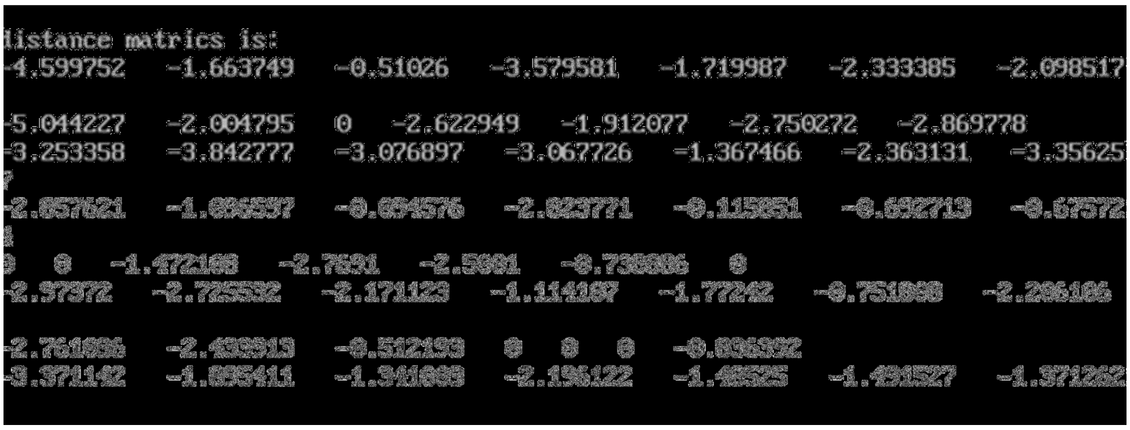


```
matrics with optimum values
1.22861   -0.231662   -0.830743   1.38346   0.234737   0.841858   0.727255
1.673085   0.109384   -1.341003   0.426827   0.426827   1.258745   1.498516
-0.117784   1.947365   1.735894   0.871604   -0.117784   0.871604   1.984995
-0.513521   -0.798814   -1.246427   0.627649   -1.369399   -0.798814   -0.695538

-3.371142   -1.895411   0.131105   0.572978   1.01485   -0.75264   -1.371262
-0.397422   0.83012   0.83012   -1.082015   0.28717   -0.739719   0.834844
-0.610107   0.538502   -0.82881   -2.196122   -1.48525   -1.491527   -0.534871

-3.371142   -1.895411   -1.341003   -2.196122   -1.48525   -1.491527   -1.371262
```

Then after we calculate the distance matrix from the given equation 9

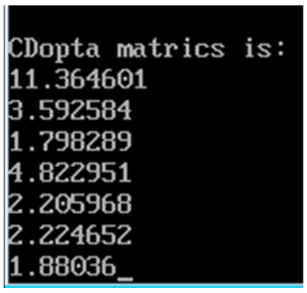


```
istance matrics is:
-4.599752   -1.663749   -0.51026   -3.579581   -1.719987   -2.333385   -2.098517

-5.044227   -2.004795   0   -2.622949   -1.912077   -2.750272   -2.869778
-3.253358   -3.842777   -3.076897   -3.067726   -1.367466   -2.363131   -3.35625
7
-2.857621   -1.696537   -0.694576   -2.823771   -0.115851   -0.692713   -0.67572
1
0   0   -1.472168   -2.7631   -2.5001   -0.738806   0
-2.97372   -2.725532   -2.171123   -1.114187   -1.77242   -0.751898   -2.206186

-2.761896   -2.439913   -0.512193   0   0   0   -0.896392
-3.371142   -1.895411   -1.341003   -2.196122   -1.48525   -1.491527   -1.371262
```

Calculate the Euclidean composite distance between each alternative Software Metrics to the optimal state OPTIMAL is derived from the equation 10



```
CDopta matrics is:
11.364601
3.592584
1.798289
4.822951
2.205968
2.224652
1.88036_
```

	Composition distance value	Rank
CC	11.364601	7
FD	3.592584	5
MTTF	1.798289	1
SDC	4.822951	6
RC	2.205968	3
CH	2.224652	4
Criteria	1.88.26	2

Table 5 Result Table

The selection of software engineering metrics have been provided in terms of the signification of the impact of software reliability .The optimal values (lowest values) show for the better selection. The Table 5 show that MTTF has been lowest value so it is best solution because it scored very low value for their criteria namely cost repeatability and relevance to reliability.

Chapter Six: **CONCLUSION AND FUTURE WORK**

The study was conducted to rank the software engineering metrics using the state-of-art knowledge in the field of software engineering. In particular, a distance based approach has been developed. It is established that once a complete set of criteria and software engineering metrics have been identified, their important weights and ratings are assigned using criteria matrix using expert elicitation, and then this method can be applied for their selection.

Software engineering metric are defined an efficient rationalization process around multi attribute decision process DBA method can be applied. These rank criteria allow a decision maker to perform not just a general analysis, but also other various focused analyses regarding his or her personal preference. 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 selection criteria have been given due consideration in the matrix method and since criteria matrix is used; the situation of indeterminacy does not arise.
- The use of distance based theory improves the decision-making procedure by considering the vagueness and ambiguity prevalent in real-world system. We also found that the use of criteria matrix 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 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.

Chapter Seven: **REFERENCES**

1. **Musa, JD, Iannino, A and Okumoto, K.** *Software Reliability: Measurement, Prediction, Applications.* New York : McGraw-Hill, 1987.
2. *A ranking of software engineering measures based on expert opinion.* **Li, M and Smidts, CS.** 2003, IEEE Transactions on Software Engineering, pp. 29(9):811–824. DOI: 10.1109/TSE.2003.1232286.
3. *Performance Analysis of Software Reliability Models using Matrix Method.* **Garg, RajPal, Sharma, Kapil and Garg, R. K.** June 2010, IEEE Transactions on reliability.
4. *Existing model metrics and relations to model quality.* **Parastoo, M and Dehlen, V.** Vancouver, BC, Canada : s.n., May 2009. ICSE Workshop on Software Quality.
5. *The state of metrics in software industry.* **Ordonez, JM and Haddad, HM.** Las Vegas, NV : s.n., April 2008. Information Technology: New Generations, 2008. ITNG 2008. Fifth International Conference. Vol. DOI: 10.1109/ITNG.2008.106, pp. 453–458.
6. **Fenton, E. Norman .** Software metrics: A rigorous approach. s.l. : Chapman & Hall, 1991.
7. **Pressman,, R S.** . *"Software Engineering A Practitioner's Approach",* . [ed.] 5th Edition.
8. *Evaluation experiments on the detection of programming patterns using software metrics.* **Kontogiannis, K.** s.l. : IEEE Computer Society Press, 1997.
9. power software. [Online] http://www.powersoftware.com/.
10. charismatak software. [Online] http://www.charismatek.com.au/.
11. software engineering institute. [Online] www.sei.cmu.edu.
12. quality managment system. [Online] http://www.iso.ch/iso/en/iso9000-14000/.
13. software metrics practies. [Online] http://www.totalmetrics.com/.
14. david consutling group. [Online] http://www.davidconsultinggroup.com.
15. *http://education.dewsoftoverseas.com/QE/QUickReference/Software%20Enginering/7.2.asp.* [Online] June 17, 2014. http://education.dewsoftoverseas.com/QE/QUickReference/Software%20Enginering/7.2.asp.
16. *Kazuhiko I. Fault generation model and mental stress effect analysis. The Journal of Systems and Software ; 26(1):. DOI: 10.1016/0164-1212(94)90093-0.* **Furuyama , T.** 1994, pp. 31–42.
17. *An analysis of factors affecting software reliability.* **Zhang , X and Pham, H.** 2000, The Journal of Systems and Software , Vols. 10.1016/S0164-1212(99)00075-8, pp. 43–56.
18. *A critique of software defect prediction models.* **Fenton, NE and Neil , M.** 25(5), 1999, IEEE Transactions Software Engineering , pp. 675–689. 10.1109/32.815326.
19. *Objective software quality assessment* . **Johnson , GL and Yu , X.** 842911., WA, U.S.A. : s.n., 1999, Proceeding of Nuclear Science Symposium (NSS),,Seattle, Vol. 10.1109/NSSMIC, pp. 1691–1698.

20. *Validation of a methodology for assessing software reliability.* **Li, M, et al., et al.** Saint- Malo, Bretagne, France : s.n., 2004. Software Reliability Engineering 15th International Symposium. pp. 66–76.
21. **Sharma , A, Kumar , R and Grover , PS.** Estimation of quality for software components: An empirical approach. ACM. *An empirical approach.ACM SIGSOFT Software Engineering Notes.* 2008, Vol. 33(6), pp. 1–10.
22. *Estimating software costs.* **Putnam , LH and Fitzsimmons, A.** s.l. : 25(11), 1979, Datamation , pp. 171–178.
23. *DESMET: A methodology for evaluating software engineering methods and tools.* **Kitchenham, B, Linkman, S and Law , D.** s.l. : 8(3), 1997, Computing and Control Engineering Journal, pp. 120–126.
24. *An instrument for measuring the key factors of success in software process improvement :.* **Dyba, T.** s.l. : 10.1023/A, 5 2000, . Empirical Software Engineering , pp. 357–390. 1009800404137.
25. *Subjective evaluation as a tool for learning from software project success.* **Wohlin , C, et al., et al.** 42(14), 2000, pp. 983–992. 10.1016/S0950-5849(00)00150-6.
26. *An experimental study of individual subjective effort estimations and combinations of the estimates.* **Host , M and Wohlin , C.** Kyoto,Japan : s.n., 1998. Proceeding of the 20th International Conference on Software Engineering. pp. 332–339. 10.1109/ICSE.1998.671386.
27. *Assessing the cost-effectiveness of inspections by combining project data and expert opinion.* **Briand , LC, Freimut , B and Vollei , F.** San Jose,CA, U.S.A : s.n., 2000. Proceedings of the 11th International Symposium on Software Reliability Engineering. Vol. 23.
28. *A methodology for ranking of software reliability measures.* **Singh , R, Singh , O and Singh , Y.** 2006, IE (I) Journal-CP , Vol. 87, pp. 14–20.
29. *A fuzzy multi-criteria decision making method for technology transfer strategy selection in biotechnology.; :. DOI:.* **Chang , PL and Chen , YC.** 1994, Fuzzy Sets and Systems , Vol. 63, pp. 131–139. 10.1016/0165-0114(94)90344-1..
30. **Goel, L A.** Software reliability models: assumption, limitations, and applicability. s.l. : IEEE Trans. Softw. Engineering, December 1985, pp. 1411–1423.
31. **Ghaly, Abdel, Chan, Y P and Littlewood, B.** Evaluation of competing software reliability predictions. s.l. : IEEE Trans. Softw. Engineering, September 1986, Vols. SE-12, pp. 950–967.
32. **Khoshgoftaar, T. M.** On model selection in software reliability. s.l. : 8th Symposium in Computational Statistics, August 1988, pp. 13–14.
33. *Software reliability model selection.* **Khoshgoftaar, T M and Woodcock, T.** s.l. : IEEE Computer Society Press, 1991. 2nd International Symposium on Softw. Reliability Engineering. pp. 183–191.
34. *A simulation study of the performance of the Akaike information criterion for the selection of software reliability growth models.* **Khoshgoftaar , T M and Woodcock, T G.** s.l. : 27th Annual Southast Region ACM Conf., April 1989 . pp. 419–423.

35. *IEEE Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software.* IEEE. New York : s.n., 1988. IEEE Std 982.2.
36. **Lawrence, JD, et al., et al.** *Assessment of software reliability measurement methods for use in probabilistic risk assessment.* Fission Energy and Systems Safety Program, Lawrence Livermore National Laboratory. s.l. : Technical Report UCRLID-136035, 1998.
37. Optimal selection and accuracy estimation of software reliability models. *PhD Thesis submitted to M. D. University.* Rohtak, India : s.n., 2011.
38. *Ranking of software engineering metrics by fuzzy basd matrix methodology.* **Garg, R K, et al., et al.** s.l. : software testing verificaiton relianing, 2013, wiley online library, Vol. 23, pp. 149-168.
39. *http://ecomputernotes.com/software-engineering/software-metrics.* [Online] June 17, 2014. http://ecomputernotes.com/software-engineering/software-metrics.
40. *Software reliability: Status and perspectives.* **Ramamoorthy , CV and Bastani , FB.** 1982, IEEE Transactions Software Engineering, pp. 354–371.
41. *Rainer RK Jr. Factors that impact implementing a system development methodology.* **Roberts, Jr TL, Gibson , ML and Fields , KT.** s.l. : IEEE Transactions on Software Engineering, 1998, pp. 640–648.
42. *A model-based framework for the integration of software metrics. .* **Evanco , WM and Lacovara , R.** 1994, The Journal of Systems Software, pp. 77–86.
43. *. Distributed computing environments: Effects on software maintenance difficulty.* **Schneberger, SL.** 1997, The Journal of Systems and Software , pp. 101–116.
44. *Software reliability metrics selecting method based on analytic hierarchy process. Quality Software, ., , 2006; . DOI:.* **Li , H, Lu, M and Li, Q.** Beijing : s.n., 2006. QSIC 2006. Sixth International Conference. pp. 337–346. 10.1109/QSIC.2006.59.
45. *Tool steel materials selection under fuzzy environment.* **Wang, MJJ and Chang , TC.** 1995, Fuzzy Sets and Systems , Vol. 72, pp. 263–270. 10.1016/0165-0114(94)00289.