Vol 4, Issue 6, June 2024 www.ijesti.com E-ISSN: 2582-9734
International Journal of Engineering, Science, Technology and Innovation (IJESTI)

LIFE CYCLE OF SOFTWARE DEVELOPMENT MAINTAINABILITY ATTRIBUTES MODEL USING FUZZIFICATION

Raman Kumar [1], Chhatarpal [2]

[1] M. Tech in Computer Science Engineering
[2] Assistant Professor, Department of Computer Science Engineering

[1,2] BM Group of Institutions, Farrukh Nagar, Gurugram.
Corresponding Author Email: ramankgg2016@gmail.com

ABSTRACT

Software Development goes through a number of phases. These phases together make a Life Cycle of Software Development. It is estimated that more that there are more than 100 billion lines of code in production in the world. As much as 80% of it is unstructured and not well documented. Maintenance can lessen these problems. Maintainability is the ability to keep the system up to date after deploy to the customer site. We studied a number of software maintainability measurement metrics and also new proposed techniques. In our research we focused on how to measure the software. After considering these factors we can conclude that how much software is maintainable This means that how the maintenance cost can be reduced and how much efforts will be required to reduce the cost. So, we will use a fuzzy logic to implement these factors. We found that fuzzy logic can be used to model uncertainty for these factors. Fuzzy logic is a way to deal with reasoning that is approximate rather than precise. Then by fuzzy logic we measured the maintainability. In our research, we considered experimental data. First, we applied these factors on data and then by fuzzy logic we measured the maintainability. This work based on Rule Base consists of number of rules. Rule structure is like “If this and/or If this than this.

Keywords: Life Cycle of Software Development, lines of code, Fuzzy Logic.

1. INTRODUCTION

This section introduces software management and its complex definitions, management templates, Fuzzy logic and the description of the various parameter forms. More in this chapter is a summary of the maintenance methods, as well as several modern strategies for measures of upkeep, such as McCabe's Cyclomatic Complexity and K.K Aggarwal. Maintenance is the concept most commonly correlated with more robust infrastructure and considerably lower long-term costs. Description of maintenance commonly find, for example:

"The effort required to fix errors, boost efficiency or other attributes after delivery of a software program or component, The climate has been modified.

Or "A program may be sustained if only minimal efforts are made to fix minor bugs." Perhaps too easy, naturally. The second is particularly misleading, since in reality it is a tautology rather than a concept. The response would be "its correction needs little effort if you wondered what a minor bug is." In addition to this very native description, Specific metric methods aim to characterize main tenability as conformity with a collection of rules that suit the observable characteristics of the system, such as high consistency, minimal coupling, etc. The key concern is the absence of clear rationale for the chosen requirements, which sometimes appear to address some technological elegance instead of enhancing program management efficiently. We published a report in German business organization on network maintenance activities in 2003. Among the 47 interviewees, 60% said that software maintenance is regarded as an "important issue", but only 20% have carried out certain quality checks on maintenance. The maintainability testing requirements used by these 20% range from object-oriented, cyclical complexity, and a small number of lines per process, to a brief description to OMG's service-oriented and model-driven architecture.

Moreover, there is nothing in common about what is actual "maintenance", whether it can be measured and whether it can be performed.

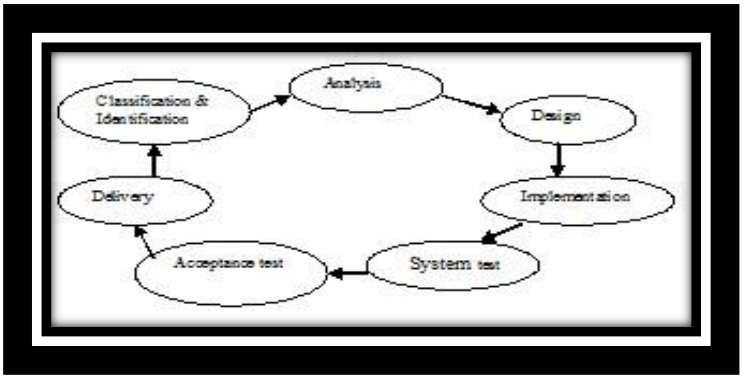


Figure 1: Maintenance Process

There is some uncertainty about "standing," whether it can be reached and whether it can be done. There are some misunderstandings. Through using "maintainability" as a concept, this ambiguity can quickly be clarified and overcome. The "fitness" finish is used to turn the "maintainable" term into a product and thereby mark it as a machine object. In addition, the term "retention" applies to the following concepts: the act of "retention" (verb) is an entity or software program that we discover. However, the assumption that the program is very unmaintainable is the characteristics of the machine, and there are many other considerations, such as training maintenance personnel, managing operational knowledge, and appropriate resources affecting software maintenance activities. Ignore this flaw is most obvious in terms of "readability".

- Computer repair resources of the company
- The system's technological capabilities under scrutiny
- Engineering criteria

Beyond skills and strategies listed above Dimension 3, technological criteria engineering plays a significant role in identifying "maintainability," as the issue of how versatile one needs at whatever points of a software system the goals are to construct and then test the software system’s versatility. For example, if your program has undergone major adjustments, the shortest practical implementation method without redesign or versatility is not just a preconceived idea of the versatility with multiple

indirectness. Maintenance costs may be lower.

1.1 Quick-fix Model

A standard approach to maintenance of software is first to focus on code and then, where necessary, make the needed adjustments. The fast-fix solution is seen in figure 2, showing the movement of changes from the old to the current iteration of the program. Ideally, the specification, research and other types of usable documentation that are influenced by the amendment will be revised once the code has been modified. Nonetheless, as a consequence of its perceived roughness, consumers tend to adjust applications quickly and cost-effectively. The software should be revised or not, because the programming is changed; time and expenditure constraint often contributes to the assumption that system adjustments are not recorded and the data is easily lost. However, the initial design will be dismantled twice, rendering it more costly to enact any reform.

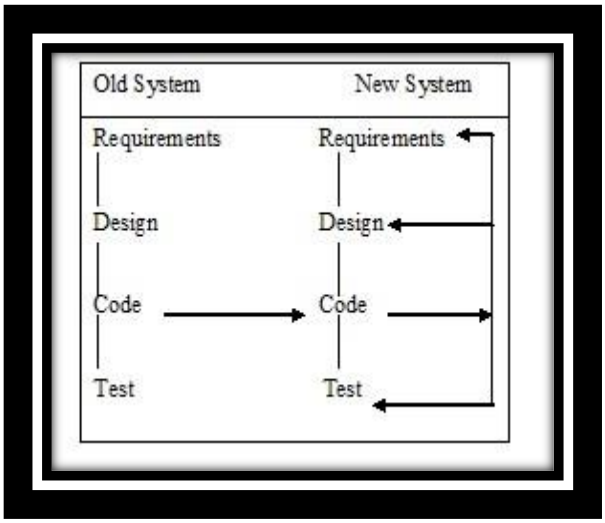


Figure 2: Systematic model

1.2 Iterative Enhancement System

Progressive life cycle models indicate a device engineering alternate solution. These models express the notion that the device criteria cannot be initially grasped and comprehended in full. Systems will then be designed in which the specifications of previous builds can be completed, updated and modified on the basis of input from users. An illustration is an iterative development that proposes structuring a problem to facilitate the creation and execution of broad solutions successively. Iterative development also describes repair as shown in Figure 3. The construction of a new construction should begin with an overview of the current infrastructure requirements, configuration, code and testing documentation, and proceed to adjust and distribute the improvements to the entire collection of documents at the highest stage. In short, the structure is revamped based on an interpretation of the current framework at every point in the evolutionary cycle. A Contact

The iterative development approach profits from preserving notes as the application shifts. Visaggio records evidence from multiple randomized trials that evaluate the rapid-fix and iterative development models and reveals that the system's functionality degrades faster for the fast-fixed model. The findings also show that companies utilizing the iterative enhancement model enable

operational improvements quicker than those using the fastfix model; this latter result is counter-intuitive, because time constraint is the most important factor to follow the fast-fix model.

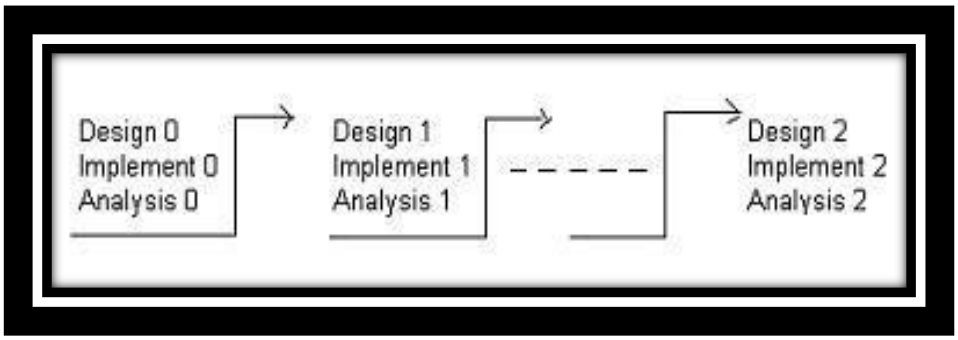


Figure 3: Iterative Improvement System

1.3 Full Reuse Prototypical

As seen in Figure 4, Basili suggests a paradigm of complete reuse that treats repair software creation that depends on reuse. This functionality for the current program and reuses the relevant specifications, architecture, programming and testing with previous systems iterations. The idea of a database and feature registry describing former iterations of the current framework and other program in the same technology domain is fundamental to a full reuse model. It specifically reuse and evidence. This also allows reusable components to be created. The iterative-enhancement approach is ideally suited to processes with a long existence and a lengthy development. In order to promote further improvements, it encourages the progression of the system. The full-use approach, by comparison, is best adapted for designing lines of complementary goods. In the short term the expense is going to be higher although, in the long run, the benefits might be sensitive; organizations following the maximum reuse model collect recycled components of all sorts and at a vast range of abstract rates.

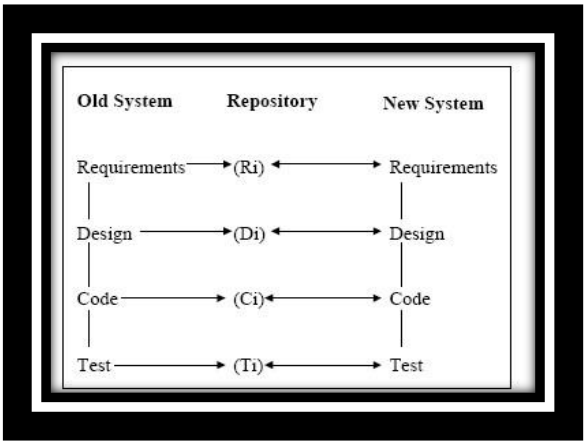


Figure 4: Full Reuse Model

Virtually every business which relies on software has a critical concern to reduce its software maintenance costs. This is no surprise because most computing systems' life cycle costs are not taken away from the creation of modern applications, but from continuously developing, modifying and addressing glitches in current apps.

1.4 Fuzzy Judgment

It is necessary to understand what fudging logic really is before I explain the processes that make fugitive logic machines operate. The value of dynamic logic comes from the assumption that the bulk of forms of human thought are provisional in nature.

The main characteristics of Zadeh Lofti's fuzzy reasoning are as follows.

- Exact reasoning, in useless logic, is regarded as a restricted case of rough reasoning
- This is a question of degree of dynamic reasoning.
- Any device that may theoretically be blurry.
- Information as a set of elastic or, similarly, a floating limit to a selection of variables was represented in fluid logic.

Inference is known as an elastic restriction propagation mechanism.

1.4.1 Why Fuzzy Logic

Fuzzy logic provides many special characteristics, which make for several control problems an especially good option.

- It is entirely stable as the dosage does not need identical noise-free inputs. Whether a feedback captive fails or is killed, it may be configured to malfunction safely. Given a number of variations, the performance control is a smooth control feature
- As the FL controller operates on user-defined control system guidelines, the device output can be adjusted and improved quickly or dramatically altered. With only correct guidelines, new sensors will quickly be incorporated into the device.
- To apply FL, Any sensor data which indicates the behavior and reaction of a device is appropriate. It helps the sensor to be inexpensive and unreliable to keep the cost and performance total of the machine minimal.
- Although the rule-based method can handle a sufficient number of inputs (1-8 or more) and produce multiple outputs (1-4 or more), it does not have enough inputs to specify the rule base. Only one step is needed to select the output, because the law that determines the relationship is required. Dividing the control scheme into smaller parts will be complicated for the operator and will require less obligation to perform many different FL controls on it.
- FL can regulate mathematically complicated or impossible nonlinear structures. It unlocks doors for control structures which are usually perceived to be automation unfeasible.

II. LITERATURE REVIEW

Shyam R. Chidamberand Chris F. Kemerer: This paper tackles these criteria by creating and applying a modern series of OO architecture metrics. Prior performance measurement work has typically been subject to one or more forms of critique, while adding to the awareness of software engineering processes in the region. Those involve the absence of a theoretical basis, inadequate generalization, or development relies too much on the calculation of suitable resources, and too much energy for selection. The Bunge ontology became the scientific basis between Wand and Weber for the OO concept approaches. Six concept indicators were developed and analytically tested against a collection of measuring criteria previously indicated. The set of requirements for program metry assessment and a short overview of the

scientific data collection sites are given for Weyuker. To order to show viability and propose an observational comparison of such interventions on two field locations, an interactive data collection method was eventually established and introduced.

Jane Huffman Hayeset. al.: The Adaptive Maintenance Effort Modell (AMEffMo) develops a concept for the calculation of adaptive program maintenance in person, for the purpose of estimating adaptive maintenance in one-to-one period. The regression trends have been effective in estimating proactive maintenance practices and the valuable knowledge for managers and maintainers.

Jane Huffman Hayes et. al. : Introduce the paradigm of observation and adoption (OMA) which assists organizations, without committing themselves and implementing largescale, enhancement of their software's creation processes. In specific, the technique was used to enhance maintenance-oriented development procedures. The idea that tech teams naturally report about issues that perform or don't operate well is based on this innovative perspective. Then, teams searched their objects and recollections for incidents to identify components of apps, procedures, measurements, etc. For the management of devices, any calculation will instead be performed to insure that the process contributes to better management. To order to meet the requires, we introduce two preventive steps, a substance preventive and expected maintenance. Also investigated were other maintenance steps that can be included in the mine phase. Finally, the team formalizes and adopts mining operations that contribute to established observations of procedures, strategies or behaviors that improve the software product. Two development ventures and a web-based healthcare infrastructure operated by a wider commercial tech company, have been experimentally analyzed in OMA.

Warren Harrison et. al.: Create a modern software maintenance model focused on an impartial judgment law that defines whether or not a specified software module can be effectively changed. The paper indicates that the early detection of shifting systems may be useful strategies for the productive distribution of maintenance capital by the usage of adjustment steps during release cycles.

Scott L. Schneberger and Ephraim R. Mclean: The paper appears to rely on two diameters of the information system design, based on the trade journal articles: the simplicity of modules and the sophistication of the structures. The bigger the machine elements, the better they are to control each other, but the tougher they are to tackle the entire program. This work was focused on a modern statistical paradigm on part numbers and variety, amount and selection, and the average pace of change for information management sophistication. For purposes addressed here, a report on the topic of the IS system and application-level developers, creators, programmers and consumer relations was undertaken on the secondary source details such as accounting costs for centralized device maintenance. The field research found that the complete sophistication of distributed systems analyzed exceeded their components' ease of use and usability. The paper also provides an outline of the evaluation and study of the new distributed computing technologies, including proposed areas of specialized work required based on the research findings and implications of the author.

H. Dieterro Mbach and Bradfordt.: Ulery has a big part in both efficiency and management issues in the nature in large-scale computing goods. Improving software maintenance needs better maintenance methodologies, better approval of product requirements before maintenance is published, and better designing methodologies to achieve the quality levels required to satisfy these parameters of approval. Systematic progress in maintenance involves awareness of current problems, capacity to change

established practices and a willingness to track their performance. Measurements in information are a tool that helps the development cycle if implemented correctly. A topdown approach to correct implementation of metrics would be required where oriented changes seek to decide what data is to be gathered and how they are to be interpreted. Within this report, a realistic solution is proposed to enhance management of applications by measurements. This method is focused on general indicators and changes templates. Models, application and functional recommendations are provided for converting them to industrial maintenance. Finally, some descriptions of implementations of the real-world management method are addressed.

George E. Stark: It describes the central role of many organizations in the maintenance of software. Those facets of products and processes which tend to influence the expense, timetable, efficiency and functionality of a software maintenance distribution are common for managers to define and calculate. This paper refers to the specific concerns of a single organization's technical service and addresses those actions on the basis of their responses. Attriving to maintain and engineering the areas of development, track progress over time, and help render choix among alternatives is assessed, both in the software maintenance phase and the resulting product.

M. Burgin et. al.: Describes the significant and fairly recent reuse of information computer systems strategy. Application. It intends to establish more technical criteria for product reusability evaluation methods and mathematical theory. After the introduction, reusability is seen as a factor of usefulness in the second part. It allows you to take advantage of expertise in software reuse metric creation and usage. The third part describes and describes various formats and levels of software indicators. The fourth part is a formal description of the software indicators and their attributes. The work focuses on the development of information engineering and, in particular, on creating more effective measures of reuse.

Melis Dagpinar and Jens H. Jahnke: A great deal of criteria for evaluating objectoriented program characteristics, such as height, shield, stability and connection, have been suggested. The findings reveal that direct coupling parameters of the scale and import are important predictors of class retention, whereas the measurements of descent, unity and indirect / export coupling are not.

Robert Lagerström Pontus Johnson: The theoretical model consists of organizations with corresponding features to construct business models of architecture. The Nose Such models offer guidance for the quantitative management review of the knowledge. In these evolving programs, IT decision makers use the model should be able to forecast the costs of evolving for specific software projects and collect risk analysis data. It encourages IT leaders to consider their project lists a focus and to prioritize reform programs.

Priyanka Dhankhar and Harish Mittal: Describe the management of applications is a measure of how easily a software program or feature can be adjusted for the purpose of fixing bugs, enhancing efficiency or other attributes or adapting to a changing setting' We provide an overview of the design of object-oriented applications in this article. They are necessary for ensuring reusability and expandability. . Through empirical review, to prove that object-oriented difficulties are usually not enough to quantify content written in other object-oriented languages, we will discuss issues such as encapsulation inheritance and text. , Information Sciences Metrics of Halstead and Cyclomatic difficulty of Mc Cabe.

Other methods for calculating: Take a variety of variables, rendering it very difficult to guess. Priyanka Dhankar and Harish MittalProvides product servicing as a function for all engineering departments as the program is shipped, configured and usable to the location of the client. The amount of time and energy taken to maintain the program in service absorbs about 40%-70% of all development cycle costs. A systematic test of object oriented computing focused on two criteria, class binding and cyclomatic complexity utilizing fused logic, is suggested in this report. In addition, this analysis provides observational evidence on class maintenance times used to test the method suggested.

Megha and Harish Mittal: The easiest way to fix errors, boost efficiency or other characteristics is to calculate software maintenance, or adjust to the changing environment, utilizing a software program or part. Maintenance relies very much on the software form, as is commonly known. Maintenance of applications is a time consuming, expensive step of the life cycle of a software system. Over the full life cycle, the time and resources needed to run software consume approximately 40% to 70% of the cost. In this article, we will discuss the manner in which a paradigm introduced decreases the uncertainty and operational costs of programs and actions. Reduces or eliminates expensive downtimes and efficient uptime improvements. At times unplanned preventive works that have less effect on development may be carried out.

Megha and Harish Mittal: Maintenance of software is a challenge undertaken by every designing party as the program is shipped, activated and usable at the customer's location. Maintenance relies very much on the software form, as is commonly known. Maintenance of applications is a time consuming, expensive step of the life cycle of a software system. This report suggests a 4-parameter comprehensive analysis for software maintenance estimation. The time invested and resources required for the management of software are roughly 40% to 70%. Using these criteria the analysis should determine how repair expenses and resources are popular. We have therefore established a blurry model for device repair measurements.

III.RESEARCH METHODOLOGY

The easiest way to change a software product is through software engineering:

- Removed defects
- Meet with modern requirements.
- Consider things easy to manage, or
- Coping in a changed climate

With other formulae measuring from the line of codes, McCabe metrics and measurements of K.K Aggarwal the main tenability index is determined. The goal of monitoring and recording maintenance is to minimize or reverse the "information entropy" or deteriorated quality pattern of a program and to determine whether it is cheaper and less costly to rewrite the application instead. Maintenance by Ambiguous Reasoning will better be measured.

Three types of resources reside in the toolbox:

- Features of the command line
- Online visualization resources
- Blocks and instances replicated

The first device group involves functions you can call from your own programs or from the command line. MATLAB M scripts, a collection of MATLAB statements, applying unique fuzzy logic formulas, are all of these features. For these functions you may show the MATLAB code with the comment

3.1. Function name sort

Through copying and renaming an M-file, you can modify how every toolbox feature function and then adjust your copy. You may also connect the M-files to the toolbox.

The GUI-based methods together offer an interface for the creation, study and execution of the fluid intervention framework.

A variety of modules are included in the Simulink simulation framework as a third type of devices. These are built especially in the Simulink setting for high-speed fuzzy logic inferences.

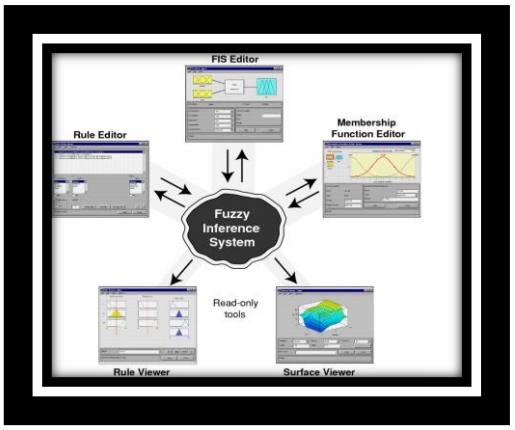


Figure 5: Fuzzy Logic Toolbox

3.2. Fuzzy Inference Arrangement

The fuzzy logic of the toolbox does not limit the number of inputs. However, due to too many inputs or too many member functions, it may be difficult to study FIS using other interface methods.

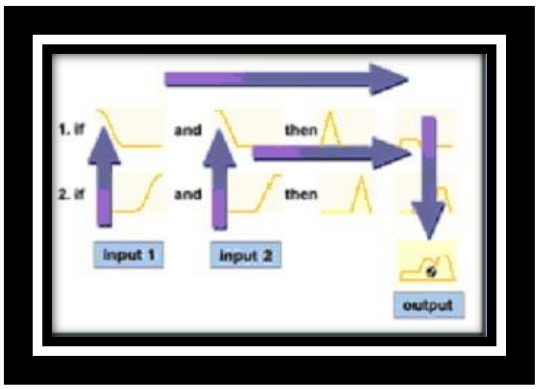


Figure 6: Fuzzy Inference Organization

Instead of editing, FIS is used for the Law Viewer and Surface Viewer. They are resources that are exclusively read only. The Rule Viewer uses the fuzzy diagram displayed at the end of the last segment on the basis of the MATLAB. Used as a treatment, it may explain the laws are active or how the outcomes are influenced by the individual participants. The Surface Viewer shows one or two of the inputs dependence of the output – that is, produces and displays the system's output surface map.

IV. PROPOSED MODEL

4.1 Introduction

Software maintenance is one job for each developing community as software is supplied, configured and operational to the customer's website. We have developed a software integrated approach using four parameters: average live variable, average life period duration, average cyclomatic complexity, average cyclomatic complexity. The software's sophistication is not calculated in the form of traditional methodologies such as code rows, Halstead Software Science Metrics, etc. We therefore have merged the conventional with the newer methodologies and established four factors. The outcome would be maintainability after processing these parameters. We present a fluffy model with the input shown in figure 7, taking two factors.

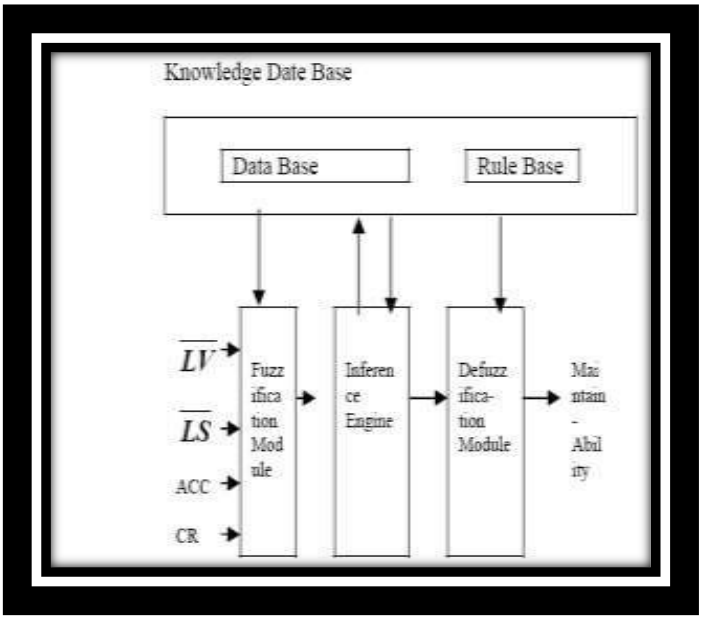


Figure 7: Fuzzy Models for Software Maintainability

This Model consists of three modules:
Fuzzification module
Fuzzy Inference Engine
Defuzzification module

4.2 Maintainability Assessment Metrics

Average number of live variable-

Real-time variables are only used when a certain number of statements are referenced before and after the statement. The activity variable is the average number of activity variables (LV) divided by the number of activity variables.

Declarations functional (n) LV = LV / n.
Where n is a statement executable.

For a system with LV = LV/m modules, m is the number of modules.

The more live variables the more complicated it is to create and sustain a program, the greater the average number.
AverageCyclomatic Complexity

McCabe describes cyclization as follows:

$$V = e – n + 2p$$

Where e = the number of corners of the system flow chart, N = node number is associated with p components.
If p = 1, then v = Π + 1, where is the number of predicates in the program.

The average complexity of the loop module is defined as the average complexity of the loop module. Such variables are used to assess device maintenance.

4.3 Fuzzy Based Maintainability Assessment

4.3.1 Fuzzification

The input will be refreshed to the table lookup or task evaluation. The research focused on 80 rules, each of which depends on how the input is parsed into a different set of Hughes languages.

Normal cycle complexity and low life cycle Normal cycle complexity, including components, short life and excellent maintenance. Before evaluating the law, we need to change the input according to each language set. For example, how many real-time variables are there on average?

Both entries can be divided into fluid sets: small, medium, and high. Medium to high. As shown in Figure 8, maintainability is divided into "large", "good", "strong", “effective", "and critical" and "failure". The results are calculated in Figures 9, 10 and 11.

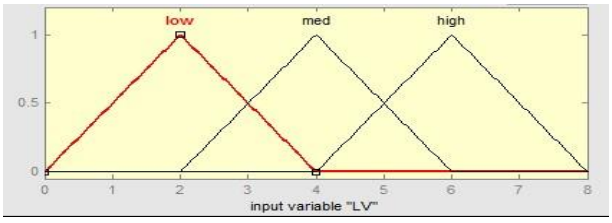


Figure 8: Fuzzification of Average Live Variable

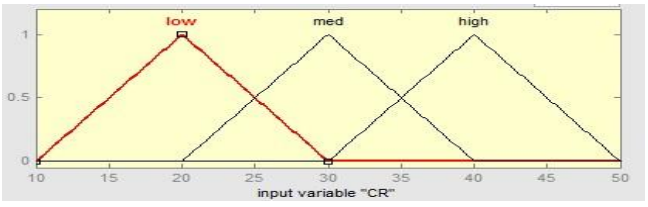


Figure 9: Fuzzification of Average life span

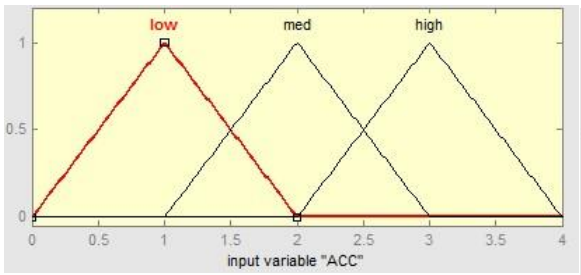


Figure 10: Fuzzification of Comment Ratio

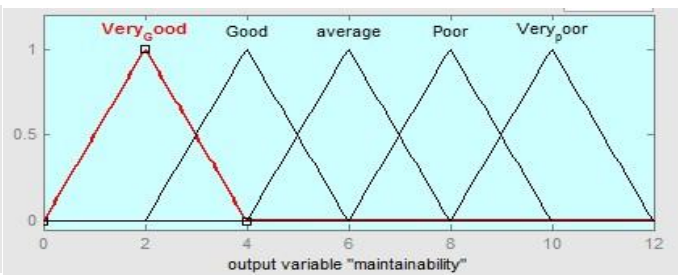


Figure 11: Fuzzification of Average Cyclomatic Complexity

Linear functions from part to part

Distribution function Gaussian

The Spiral Sigmoid

Square and cubic curves of polynomial

The simplest membership functions are generated by direct lines. The triangular component function is the smallest, and it has the name trimf function. This feature is nothing but a three-point set forming a triangle. The function of the trapezoidal member trapmf is only a truncated triangular curve with a flat top. The simplicity comes from these subscription membership apps.

4.3.2 Rule Base

We need to describe rules in Rule Base. Such laws are followed when the performance is measured. For both input and output, in the first stage, all language sets were specified. Every possible input combination leading to 3 ^ 4 i.e. 81 sets is considered. For all 81 variations, maintainability is graded by expert opinion as Very Good, Fair, Bad or Very Bad. This results in 81 rules for the fuzzy model being developed. They are all shown as follows:
(Low LV), (low LS), (low CR), (low ACC) maintainability is very good.

1. (LV is medium) and (LS is medium) and (CR is medium) and (ACC is medium) maintainability is average.
2. In the case of (High LV), (High LS), (High CR), (High ACC), the maintainability is very poor.
3. (Low LV), (Low LS), (Low CR), (Low ACC), the maintainability is very good.
4. If (low LV) and (low LS) and (medium CR) and (high ACC), the maintainability is good.
5. The maintainability of (low LV) and (medium LS) and (medium CR) and (high ACC) is very average.
6. The maintainability of (low LV) and (medium LS) and (low CR) and (low ACC) is very good.
7. When (LV is medium), (LS is medium), (CR high) and (ACC high), maintainability decreases.
8. (LV medium) and (LS medium) and (CR low) and (ACC low) maintainability is average.
9. (High LV), (Low LS), (Low CR), (Low ACC) maintainability is very good.
10. (High LV), (High LS), (Medium CR) and (Medium ACC) reduce maintainability.
11. When (LV high), (LS high), (CR high), (ACC low), maintainability is very poor.
12. The maintainability of (high LV) and (medium LS), (medium CR) and (medium ACC) is average
13. The maintainability of (Low LV) and (Low LS), (Low CR) and (Low ACC) is very good.
14. (Low LV), (Medium LS), (High CR) and (High ACC) reduce maintainability.
15. (LV is medium), (LS is high), (CR is low), (ACC is medium), maintainability is average.
16. (High LV), (Low LS), (Medium CR), (Low ACC) Good maintainability.
17. (LV is medium), (LS is low), (CR is high), (ACC is low), good maintainability.
18. (High LV) and (Medium LS) and (Low CR) (High ACC) reduce maintainability.
19. (Low LV), (High LS), (Medium CR) and (High ACC) reduce maintainability.
20. (High LV) and (low LS) and (medium CR) and (medium ACC) maintainability is average.
21. (LV is medium), (LS is high), (CR is low) and (ACC is low), good maintainability.

At 23. (High LV) and (Medium LS), (Low CR) and (ACC), the maintainability is average.

24. The maintainability of (low LV) and (medium LS), (high CR) and (low ACC) is good.
25. (Low LV), (low LS), (low CR), (high ACC) maintainability is very good.
26. For (LV Medium) and (LS Medium) and (CR Medium) and (ACC Low), maintainability is average.
27. When (LV high), (LS high), (CR high) and (ACC high), maintainability is very poor.
28. (LV is medium), (LS is medium), (CR is medium) and (ACC is high), maintainability is average

29. If (LV is high), (LS is medium), (CR is medium) and (ACC is high), the maintainability is reduced.
30. (LV is medium), (LS is low), (CR is high), (ACC is medium), maintainability is average.
31. When (LV low), (LS high), (CR high) and (ACC medium), maintainability decreases.
32. (Low LV), (high LS), (low CR), (low ACC) maintainability is very good.
33. (Low LV), (High LS), (Low CR), (Low ACC) Good maintainability.
34. (Low LV) and (medium LS) and (medium CR) and (medium ACC) maintainability is average.
35. (High LV), (Low LS), (Low CR), (Low ACC), good maintainability.
36. (High LV), (Middle LS), (High CR) and (High ACC) make maintenance very poor.
37. (Low LV), (low LS), (medium CR), (medium ACC), good maintainability.
38. Average usability of (LV med) and (LS med) and (Low CR) and (ACC med).
39. (High LV), (High LS), (Low CR), (Low ACC) reduce maintainability.
40. When (LV high), (LS low), (CR high) and (ACC high), maintainability decreases.
41. When (LV low), (LS low), (CR high) and (ACC high), maintainability decreases.
42. (LV is medium), (LS is high), (CR is high), (ACC is medium), maintainability is very average.
43. When (LV is medium) and (LS is high), (CR is high) and (ACC is high), maintainability is very poor.

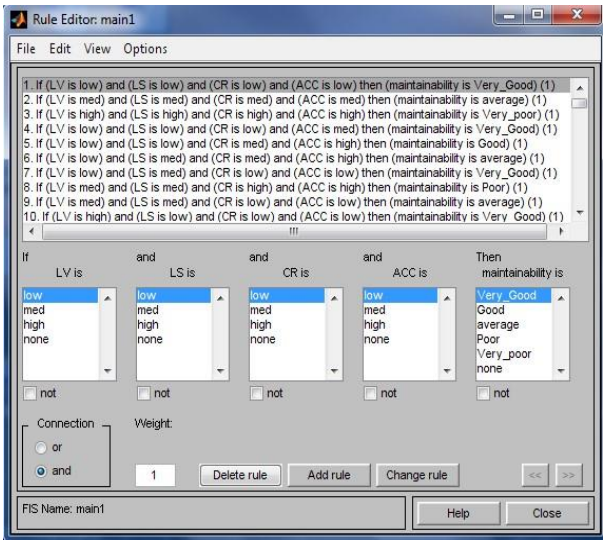


Figure 12: Rules Set

When the inputs have been fluctuated, we know the degree to which each part of the past of each law is fulfilled. If more than one part of the precedent of a given law is applied, the fuzzy operator shall obtain a number which represents the results of the precedent. This number is used for the output function. Use two or more membership values from the fuzzy input variables to access the fuzzy operator. The result is a single explanation of facts.

4.3.3 Defuzzification

The input is a set, and the output is a single number during the disinfection process. As fluorescence contributes to the rule calculation during the intermediate stages, a single number is the final desired output for each variable. Nevertheless, in order to solve an output value from the set, the sum of a fuzzy set covers a number of output values and must therefore be defuzzied.

The centroid computation which returns the central area under the curve is perhaps the most common DE fuzzifying process. Five integrated approaches are supported: centroid, bisector, maximum center, maximum width and top.

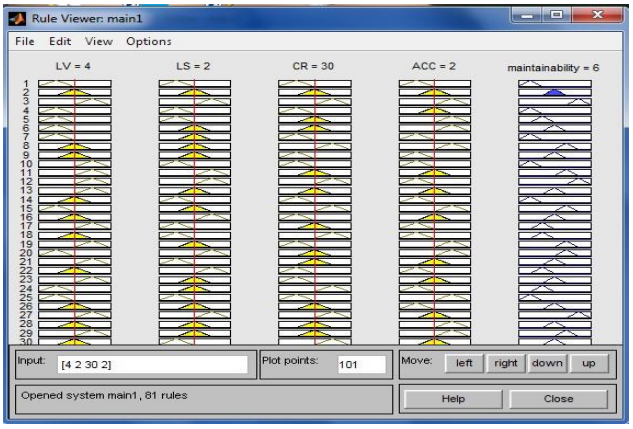
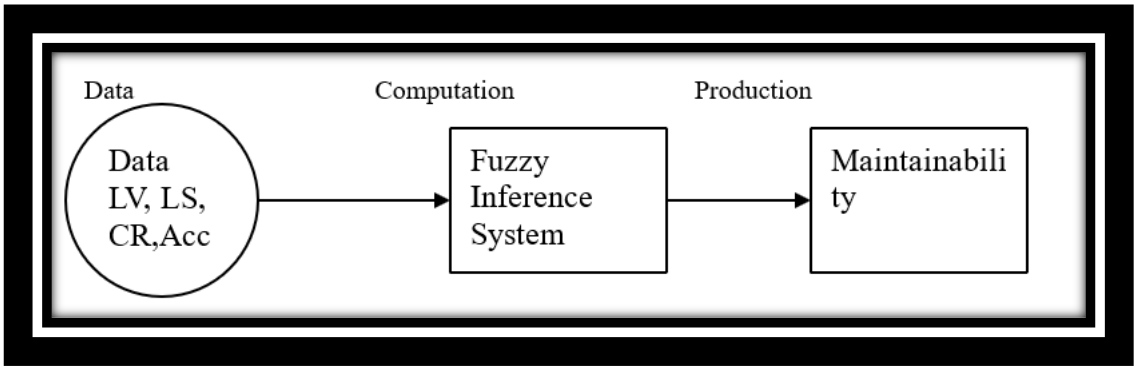


Figure 13: Defuzzification of Output

V. EXPERIMENTAL STUDY AND RESULTS

5.1. Introduction

Maintenance is generally seen to rely heavily on the type of data. We have attempted to assess device maintenance. With reasons and some empirical analysis, we have tried to show that the complexity of the program could not be calculated by commonly used complexity measures like code lines, Halsteade Program Science Metrics and others. Such indicators may not be ideal for calculating software maintenance costs and workload. We also looked at the five software ventures of students in undergraduate engineering. After the projects have been considered, the various attributes in these projects apply, the values of the individual projects in all four parameters are different. The input data is the value of the vier parameter processed or calculated by the fuzzy inference system, and the output is maintainable



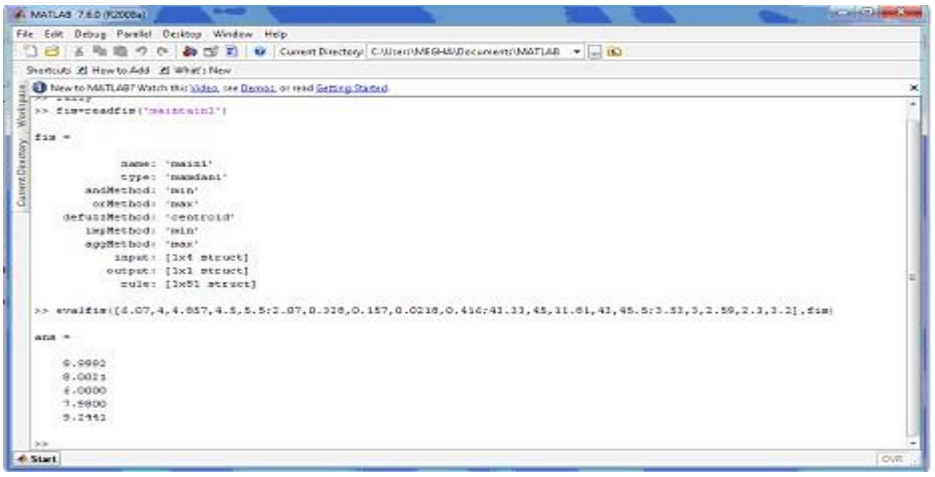


Figure 14: Snapshot of Matlab

5.2. Investigational Result

There is a variable. This variable is available. The proposed fuzzy model is also used to estimate maintenance. The results are shown in Table 1.

Table2: Value of maintainability

P.No	LV	LS	CR	ACC	Maint.
1	6.07	2.07	43.33	3.53	9.9992
2	4	0.328	45	3	8.0021
3	4.857	0.157	11.81	2.59	6.0000
4	4.5	0.0218	43	2.3	7.9800
5	5.5	0.416	45.5	3.2	9.2441

The verification diagram is also displayed on various diagrams. In these figures, all four parameters are on one axis, and the items are not on the other axis.

The co-relation between four parameters and main tenability is almost impossible. The maintenance of these four parameters is not predictable individually. The verification is done through the fuzzy model, and using the results shown in the table above, the combined values used for maintenance can also provide the best results on each input indicator.

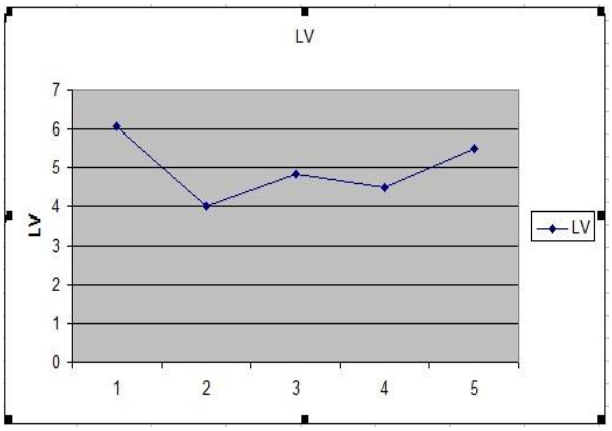


Figure 15: Diagram of Live variables

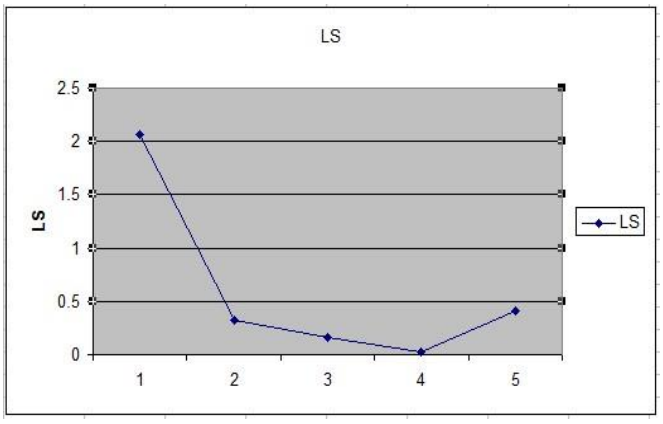


Figure 16: Diagram of Live Span

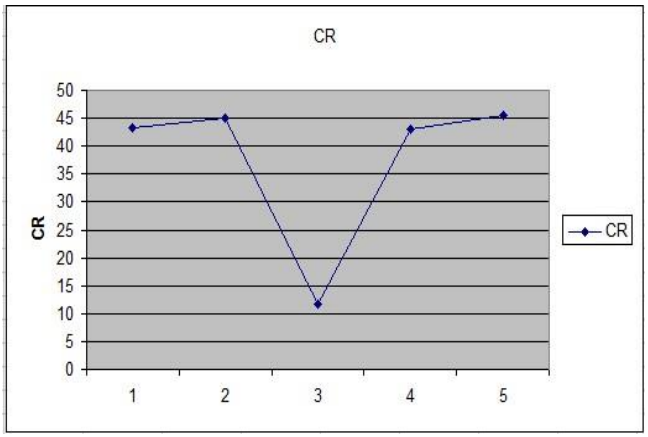


Figure 17: Diagram of Comment Ratio

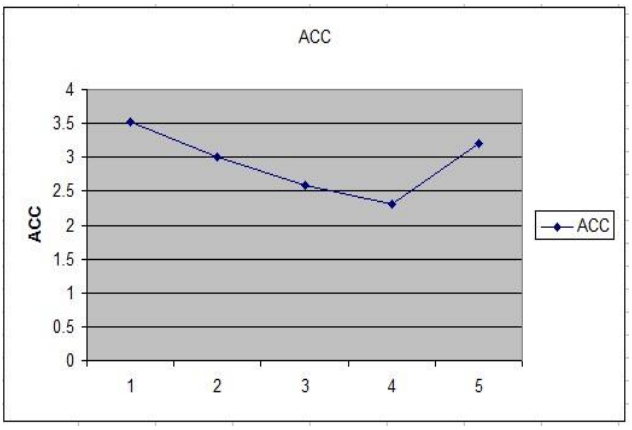


Figure 18: Diagram of Cyclomatic Complexity

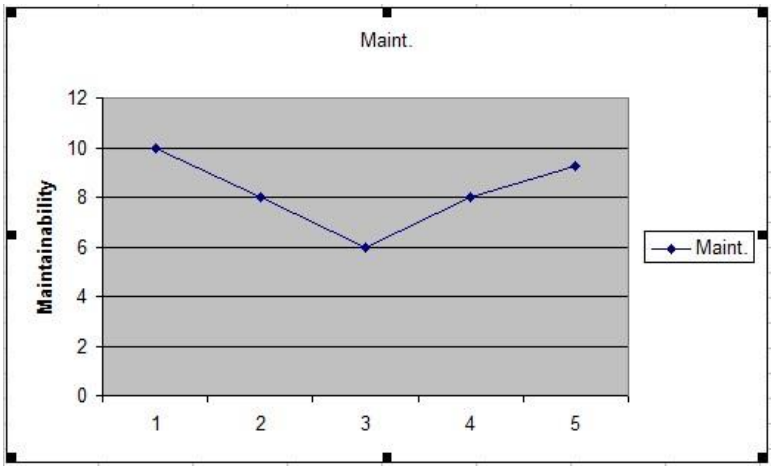


Figure 19: Diagram of Maintainability

VI. CONCLUSION AND FUTURE WORK

There are typical metrics to measure the maintenance of software measures such as code line, the information science metrics of Halstead and the cyclomatic complexity of Mc Cabe. However, these metrics cannot be used to measure software maintenance. Maintenance depends very much on the software type, as is commonly seen. We tried to demonstrate by arguments and empirical study that the software's sophistication might not be calculated by traditional metrics.

This report proposes a 4-parameter integrated analysis for software maintenance calculation. The time spent and resources required for the maintenance of software use approximately 40% to 70%. Through these parameters the study will evaluate how maintenance costs and efforts are reduced. We have therefore established a fuzzy model for software maintenance measurements.

6.1 Contribution of Present Work

Various techniques have been developed, including various major ductility factor measurement factors, such as Chandrasekhar. Consider four factors, such as the average number of real-time variables, average real-time span, and comment rate to measure the main persistence. We found that these variables provide a more detailed view of software maintenance. A fuzzy model can be used to estimate maintenance, and the results use empirical results to prove that the comprehensive maintenance value produces better results than a single input indicator.

Since different values of the four parameters are considered, the values of these parameters should be small to keep maintenance costs low. Reduce the work of calculating sustainability.

6.2 Future Scope

Further work to increase the accuracy of measurement in this field may be undertaken to build such a framework. We suggest that this model be tested in real time. The time needed to correct this error in the maintenance period is determined when any error is found in the project.

REFERENCES

1. Rikard Land Mälardalen "Software Deterioration And Maintainability – A Model Proposal" in 1995 University Department of Computer Engineer
2. Khairuddin Hashim and Elizabeth Key " A SOFTWARE MAINTAINABILITY ATTRIBUTES MODEL" Malaysian Journal of Computer Science
3. C. van Koten 1 and A.R. Gray 'An application of Bayesian network for predicting object-oriented software maintainability' in 2005 Department of Information Science,University of Otago, P.O.Box 56, Dunedin, New Zealand
4. K.K. Aggarwal et. al. 'Measurement of Software Maintainability Using a Fuzzy Model' Journal of Computer Sciences 1(4):538-542, 2005
5. P. K. Suri1, Bharat Bhushan2 "Simulator for Software Maintainability" Kurukshetra University, Kurukshetra (Haryana) India IJCSNS International Journal of Computer Science and Network Security, VOL.7 No.11, November 2007
6. Markus Pizka and Florian Deißenböck ' How to effectively define and measure maintainability' in 2007
7. Mehwish Riaz, Emilia Mendes, Ewan Tempero 'A Systematic Review of Software Maintainability Prediction and Metrics, New Zealand in2009978-1-4244-48418/09/$25.00 ©2009 IEEE
8. Priyanka Dhankhar1, Harish Mittal2 'SOFTWARE MAINTAINABILITY IN OBJECT ORIENTED SOFTWARE' in 2010 proc.conference 8th may 2010.
9. Chikako van Koten Andrew Gray An Application of Bayesian Network for Predicting Object-Oriented Software Maintainability in March 2005 ISSN 1172-6024
10. Berns, G., 1984 "Assessing Software Maintainability ."Communications of the ACM, 27: 14-23.
11. Baker, A.L.et.al. and R.W.Witty, "A Philosophy for Software Measurement," Journal of Systems and Software, 12, 277-281 (2000).
12. Wilde, N. and Ross Huitt: "Maintenance Support for Object- Oriented Programs," Proceedings of IEEE Conference on Software Maintenance Wilde, N. and Ross Huitt: "Maintenance Support for Object- Oriented Programs," Proceedings of IEEE Conference on Software Maintenance
13. Booch, G., "Object Oriented Development," IEEE Transactions on Software Engineering, SE-12, 211-221, 1986.
14. Halstead,Maurice H. "Elements of Software Science" Elsevier north Holland,New York,1997.
15. R. K. Bandi et. al. "Predicting Maintenance Performance Using Object-Oriented Design Complexity Metrics", IEEE T Software Eng, 29, 1, Jan. 2003, pp. 77 – 87.
16. Muthanna, S., K. Kontogiannis and B. Stacey, 2000. 'A maintainability model for industrial software systems using design level metrics.' Proc. Seventh Working Conf.
17. [17].Land R.,:Measurement of Software Maintainability", In Proceedings of Artes Graduate Student Conference, ARTES, 2002
18. [18]Chandershekhar Rajaraman Michael R. Lyu "Reliability and Maintainability related software metrics in C++" 2003.
19. Alain April1 et. al. "Software Maintainence Maturity Model: The software maintainence process model" 2004
20. McCabe Thomas j. "A Complexity Measure", IEEE Transaction Software Engineering,Vol2, December 1976.
21. Roger Jang et al.,1995. Fuzzy Logic Toolbox for Matlab. The Math Works,USA.
22. Shyam R. Chidamber and Chris F. Kemerer "A Metrics Suite for Object Oriented Design", June 1994.
23. Biggerstaff, T., and C. Richter, "Reusability Framework, Assessment, and Directions," IEEE Software, March 2000,pp.41-44.
24. [24] Jane Huffman Hayes "A Metrics-Based Software Maintenance Effort Model" 2000.

25. [25]Sneed, H. and A. Mercy, 1985. Automated Software Quality Assurance. IEEE Trans. Software Eng., 11Bi, 9: 909- 916.
26. George E. Stark “Measurements for Managing Software Maintenance”1996 IEEE
27. Warren Harrison, Curtis Cook “Insights on Improving the Maintenance Process through Software Measurement”1990 IEEE.George E. Stark “Measurements for Managing Software Maintenance” 1996 IEEE.
28. Chidamber, Shyam R. and Chris F. Kemerer.,"Towards a Metrics Suite for Object-Oriented Design," OOPSLA 1991.
29. B. Boehm. Software Engineering Economics. Prentice-Hall, 1981.
30. B. W. Boehm et al. Characteristics of Software Quality. North-Holland, 1978.
31. D. Coleman, D. Ash, B. Lowther, and P. W. Oman. Using metrics to evaluate software system maintainability. Computer, 27(8), 1994.
32. F. Deißenböck and M. Pizka. Concise and consistent naming. In IWPC 2005, pages 97–106, Washington, DC, USA, 2005. IEEE Computer Society.
33. F. Deißenböck, M. Pizka, and T. Seifert. Tool-supported realtime quality assessment. In Pre-Proceedings of STEP 2005, Budapest, Hungary, 2005.
34. R. G. Dromey. A model for software product quality. IEEE Trans. Softw. Eng., 21(2), 1995.
35. R. G. Dromey. Cornering the chimera. IEEE Software, 13(1), 1996.
36. N. Fenton. Software measurement: A necessary scientific basis. IEEE Tran. Soft. Eng., 1994
37. M. Halstead. Elements of Software Science. Elsevier Science Inc., New York, NY, USA, 1977.
38. C. S. Hartzman, C. F. Austin. Maintenance productivity. In CASCON 1993. IBM Press, 1993.
39. IEEE 1219 Software maintenance. Standard, IEEE, 1998. ISO 9126-1 Software engineering - Product quality - Part 1: Quality model. ISO, 2003.
40. C. Kaner and W. P. Bond. Software engineering metrics: What do they measure and how do we know? In METRICS 2004. IEEE CS Press, 2004.
41. K. Katheder. Studie zur Software-Wartung. Bachelor thesis, TU München, Germany, 2003.
42. B. Kitchenham and S. L. Pfleeger. Software quality: The elusive target. IEEESoftware,1996.
43. R. Marinescu and D. Ratiu. Quantifying the quality of object-oriented design: The factor-strategy model. In WCRE 2004. IEEE CS Press, 2004.
44. J. McCall and G. Walters. Factors in Software Quality. The National Technical Information Service (NTIS), Springfield, VA, USA, 1977.
45. Megha and Dr. Harish Mittal “Review of Software Maintainability Prediction Modeling” proceeding of 1st National Conference on Advances Computational Intelligence NCACI-2011 9th July 2011.
46. Megha and Dr. Harish Mittal “Fuzzy Based Software Maintainability Prediction Modeling” proceeding of 1st National Conference on Advances Computational Intelligence NCACI-2011 9th July 2011.
47. M. Paulk, C. V. Weber, B. Curtis, and M. B. Chrissis. The Capability Maturity Model: Guidelines for Improving the Software Process. Addison-Wesley, 1995.
48. T. M. Pigoski. Practical Software Maintenance. Wiley Computer Publishing, 1996.
49. STSC. Software Reengineering Assessment Handbook v3.0. Technical report, STSC, U.S. Department of Defense, Mar. 1997.
50. J. Q. Wilson and G. L. Kelling. Broken windows. The Atlantic Monthly, 249(3), 1982.
51. B. Wix and H. Balzert, editors. Softwarewartung. Angewandte Informatik. BI Wissenschaftsverlag, 1988.
52. M. Fowler. Who needs an Architect? IEEE Software, pages 11-13, 20(5), September 2003.
53. F. Deißenböck, S. Wagner, M. Pizka, S. Teuchert, J.-F. Girard. An Activity-Based Quality Model for Maintainability. Submitted to Int. Conf. on Software Maintenance, Paris, 2007.
54. M. Pizka. Code Normal Forms. NASA SEW-29. Greenbelt, MD. April, 2005.
55. M. Broy, F. Deißenböck, M. Pizka. Demystifying Maitainability. Proceedings of the 4th Workshop on Software Quality. ACM Press. Shanghai, China. 2006