# Average Span & Comment Ratio for Maintenance of Model Using Fuzzy Technique

***Rupali Malhotra & **Naveen Jindal**
*Head, Department of Computer Sciences & Engineering, Sat Priya Group of Institutions, Rohtak, Haryana
** M.Tech. Student, Department of Computer Sciences & Engineering, Sat Priya Group of Institutions, Rohtak, Haryana
(Email: jindalnj2012@gmail.com)

## Abstract

Various metrics-based approaches try to define maintainability as compliances to a set of rules that correspond to measurable properties of the code, such as strong cohesion, limited coupling etc. The general problem with this approach is the lack of a sound rationale for the selected criteria which in turn sometimes has a tendency of discussing some kind of technical beauty instead of effectively improving software maintenance. We measured the maintainability of the software. The statement is "Software Maintainability Prediction Modeling". Following Steps showed the research methodology-First step is to measure Software Maintainability metrics-Average Life Variable span and Comment Ratio. We studied the different value of the four parameters, the value of these parameters should be low so that the maintainability will be low and the maintenance cost will be reduced. The efforts required for measuring the maintainability will be reduced. Minimizes or eliminates costly downtime increases profitable uptime. Fuzzy model helps a lot to validate these attributes. Future work that can be done in this field to improve the accuracy of measurement, so as such system can be developed. We suggest the validation of this model in real time. When some error will be introduced in the project the time taken for correction of this error in maintenance time will be calculated.

*Key Words: Fuzzy Technique, Average Span, Comment Ratio, Maintainability*

## Introduction

The term most frequently associated with more flexible software and significantly reduced long-term costs is maintainability. Frequently found definition of maintainability such as:

Besides this rather native definition, various metrics-based approaches try to define maintainability as compliances to a set of rules that correspond to measurable properties of the code, such as strong cohesion, limited coupling, etc. The general problem with this approach is the lack of a sound rationale for the selected criteria which in turn sometimes has a tendency of discussing some kind of technical beauty instead of effectively improving software maintenance. In 2003, a study on software maintenance practices in German software organization. While 60 % out of the 47 respondents said that they would consider software maintenance as a “significant problem”, only 20% performed specific checking for maintainability during quality assurance. The criteria used by these 20% to check for maintainability differed significantly and ranged from object orientation, cyclomatic complexity  limited numbers of lines per method, descriptive identifier naming, down to service oriented architecture or OMG’s model driven architecture. Hence, there is little common ground on what “maintainability” actually is, how it can be assessed and how it could be achieved. A maintenance process is shown in **Figure-1**.

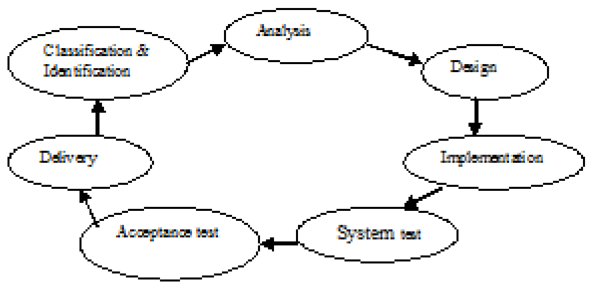


**Figure-1: Maintenance Process**

There is a little confusion on what “maintainability” actually is, how it can be accessed, how it could be achieved. This confusion can easily be explained and resolved by considering “maintainability” as a term. The “ility” ending is used to transform the adjective “maintainable” into a noun and thereby denote it as a property of a system. The adjective “maintainable” in turn denotes the assumption that the activity “to maintain” (verb) is a property of the object that we regard, i.e. a software system. However, the perception that the ability to maintain a system was a property of the system is very limited and neglects various other factors that have a strong influence on software maintenance activities, such as the qualification of maintainers, organizational knowledge management and adequate tools. This shortcoming is most obvious when it comes to “readability” since the ability to read is primarily not a property of the document or program to read but a question of the skills reader.
Therefore, we strongly argue that maintainability is not solely a property of a system but touches three different dimensions:

1. The skills of the organization performing software maintenance
2. Technical properties of the system under consideration

3. Requirements engineering

## Types of Maintenance

Accordingly, several authors consider a fourth category of maintenance, named preventive maintenance, which includes all the modifications made to a piece of software to make it more maintainable.

- Corrective Maintenance: Reactive modification of a software product performed after delivery to correct discovered problems. It deals with fixing bugs in the code.
- Adaptive Maintenance: Modification of a software product performed after delivery to keep a software product usable in a changed or changing environment. It deals with adapting the software to new environment.
- Perfective Maintenance: Modification of a software product performed after delivery to improve performance or maintainability. It deals with updating the software according to changes in user requirements.
- Preventive Maintenance: Modification of a software product performed after delivery to detect and correct latent faults in the software product before they become effective faults. It deals with updating documentation and making the software more maintainable.

These four types of maintenance can characterize all changes to the system. Corrective maintenance is-: '*Traditional maintenance'* while other types are considered as-
*'Software evolution'* Maintenance is needed to ensure that the software continues to satisfy user requirements.

- Correct Faults
- Improve the design
- Implement enhancements
- Interface with other system
- Migrate legacy software
- Retire software

## Literature Review

**Aggarwal et. al.** described that Software maintenance is a task that every development group has to face when the software is delivered to the customers' site, installed and is operational. The time spent and effort required for keeping software operational consumes about 40-70% of cost of entire life cycle. This study proposes a four parameter integrated measure of software maintainability using a fuzzy model. The study also includes empirical data of maintenance time of projects which has been used to validate the proposed model.

**Alain** addressed the assessment and improvement of the software maintenance function by proposing improvements to the software maintenance standards and introducing a proposed maturity model for daily software maintenance activities: Software Maintenance Maturity Model (SMmm). The software maintenance function suffers from a scarcity of management models to facilitate its evaluation, management, and continuous improvement. The SMmm addresses the unique activities software maintenance while preserving a structure similar to that of the CMM maturity model. It is designed to be used as a complement to this model. The SMmm is based on practitioners experience, international standards, and the seminal literature on software maintenance. We present the model's purpose, scope, foundation, and architecture, followed by its initial validation.

**Dhankhar and Mittal** described Software maintainability is a measure of the ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment' It is generally seen that maintainability very much depends on the type of Programs. In this Paper, we include an analytical study for maintainability of Object Oriented Software's. Inheritance and polymorphism are key concepts in object oriented programming (OOP), and are essential for achieving reusability and extendibility, but they also make programs more difficult to understand. .We have tried to show by arguments and by some empirical analysis that widely used complexity metrics like lines of code, Halstead's Software Science Metrics and Mc Cabe's Cyclomatic Complexity may not be appropriate to measure the complexity of Object Oriented programmed those written in other object oriented languages, since they do not address concepts like inheritance and encapsulation. Some other measuring techniques take a number of factors, which makes estimation very complex.

**Megha and Mittal** studied Software maintainability is a measure of the ease with which a software System or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment. It is generally seen that maintainability very much depends on the type of Programs. Software maintenance is a time consuming and expensive phase of a software product's life cycle. The time spent and effort required for keeping software operational consumes about 40-70 % of cost of entire life cycle. In this Paper, we will focus on how a proposed model will reduced the complexity of the projects and their maintenance cost and efforts also. Minimizes or eliminates costly downtime increases profitable uptime. Reduces unscheduled maintenance repairs can be made at times that least affect production.

**Megha and Mittal** emphasized that Software maintenance is a task that every development group has to face when the software is delivered to the customer's site, installed and is operational. It is generally seen that maintainability very much depends on the type of Programs. Software maintenance is a time consuming and expensive phase of a software product's life cycle. The time spent and efforts required for keeping software operational consumes about 40-70% of cost of the entire life cycle This study proposes a four parameters that integrated to measure the software maintainability. This study will evaluate how to reduce the maintenance cost and the efforts by using these parameters. So, we have developed fuzzy based model for measuring the software maintainability.

## Research Methodology

### Matlab Fuzzy Logic Tool Box

The Fuzzy Logic Toolbox is a collection of functions built on the MATLAB numeric computing environment. It provides tools for you to create and edit fuzzy inference systems within the framework of MATLAB. This toolbox relies heavily on graphical user interface (GUI) tools to help you accomplish your work, although you can work entirely from the command line if you prefer.

The toolbox provides three categories of tools:

- Command line functions
- Graphical interactive tools
- Simulink blocks and examples

The first category of tools is made up of functions that you can call from the command line or from your own applications. Many of these functions are MATLAB M-files, series of MATLAB statements that implement specialized fuzzy logic algorithms. You can view the MATLAB code for these functions using the statement type function_ name

You can change the way any toolbox function works by copying and renaming the M-file, then modifying your copy. You can also extend the toolbox by adding you're M-files. Secondly, the toolbox provides a number of interactive tools that let you access many of the functions through a GUI. Together, the GUI-based tools provide an environment for fuzzy interference system design, analysis, and implementation. The third category of tools is a set of blocks for use with the Simulink simulation software. These are specially designed for high speed fuzzy logic inference in the Simulink environment.

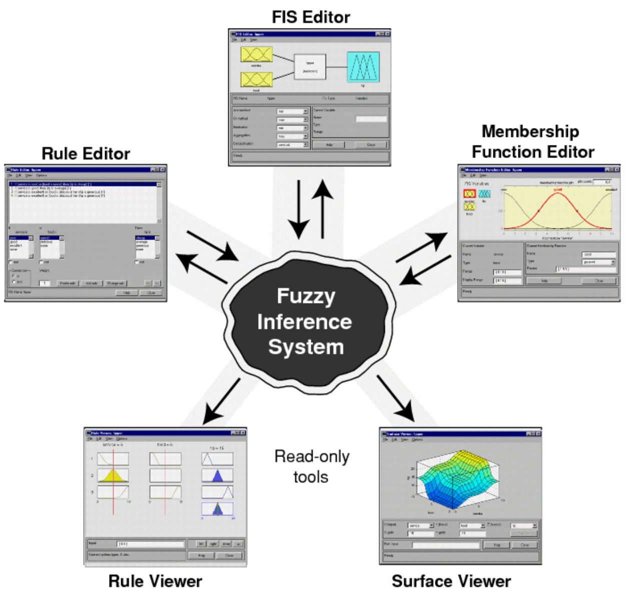


**Figure-2: Fuzzy Logic Toolbox**

## Fuzzy Inference System

The FIS Editor handles the high-level issues for the system: How many inputs and output variables? What are their names? The Fuzzy Logic Toolbox doesn't limit the number of inputs. However, the number of inputs is too large, or the number of membership functions is too big, then it may also be difficult to analyze the FIS using the other GUI tools. The Membership Function Editor is used to define the shapes of all the membership functions associated with each variable.

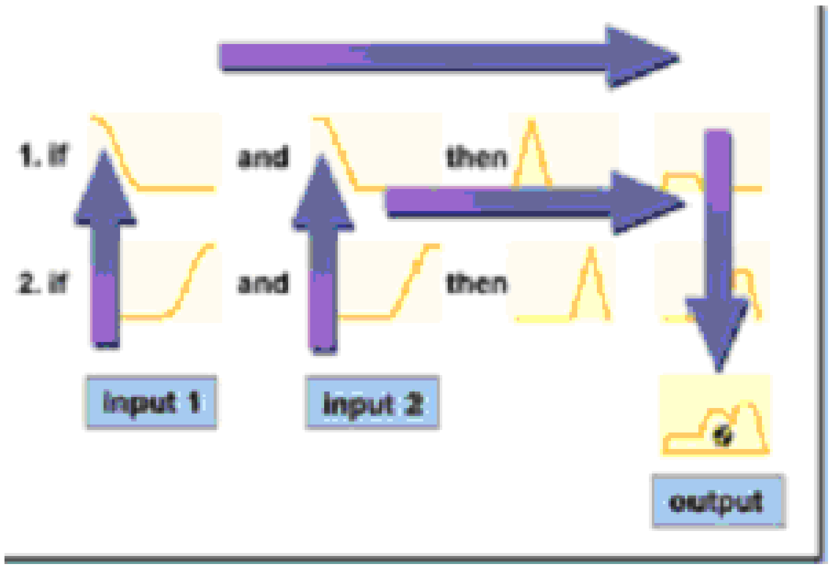


**Figure-3: Fuzzy Inference System**

The Rule Editor is for editing the list of rules that defines the behavior of the system. The Rule Viewer and the Surface Viewer are used for looking at, as opposed to editing, the FIS. They are strictly read-only tools. The Rule Viewer is a MATLAB based display of the fuzzy inference diagram shown at the end of the last section. Used as a diagnostic, it can show which rules are active, or how individual membership function shapes are influencing the results. The Surface Viewer is used to display the dependency of one of the outputs on any one or two of the inputs- that is, it generates and plots an output surface map for the system.

## Problem Statement

We measure the maintainability of the software. The statement is "Software Maintainability Prediction Modeling". Following Steps shows the research methodology-First step is to measure Software Maintainability metrics-

Average Life Variable span

Comment Ratio

Second step is applying these estimated metrics to FIS (Fuzzy Inference System) tool to get maintainability measure.
Third step is generating the experimental data of these attributes.

## Proposed Model

Average Number of Live Variable-
Average Life Variable Span-

Average life variable span**:** The span is the number of statements between two successive references of the some variable. The average span size (LS) for a program could be completed using the equation.

$$LS\ program = LS/n$$

Where n is an executable statement.

**Comment Ratio-**

Comment ratio is defined as

$$CR = (s+c)/c$$

Where s denotes total lines of code and c represents total number of comment lines. The lower the ratio, the better is the readability, and the better the readability, the better is the

maintainability. Comments provide better readability and therefore the Comments Ratio is an important factor that affects maintainability.
These factors will be used to measure the maintainability of software.

## Fuzzy Based Maintainability Assessment

### Fuzzification

The first step is to take the inputs and determine the degree to which they belong to each of the appropriate fuzzy sets via membership's functions. In Fuzzy Logic Toolbox, the input is always a crisp numerical value limited to the universe of discourse of the input variable (in this case the interval between 0 and 10) and the output is a fuzzy degree of membership in the qualifying linguistic set (always the interval between 0 and 1). Fuzzification of the input amount is to either a table lookup or a function evaluation.
Our study based on eighty rules, and each of the rules depends on resolving the inputs into a number of different fuzzy linguistic sets: If Comment Ratio is low and Average Cyclomatic Complexity is low and Live Variable is low and Life Span is low then maintainability is very good and so on. Before the rules can be evaluated, the inputs must be fuzzified according to each of these linguistic sets. For example, how much Average number of live variables is?
All these inputs can be classified into fuzzy sets viz. Low, Medium, and High as shown in **Figures-4 & 5.** The output maintainability is classified as Very Good, Good, Average, Poor, Very poor as shown in **Figure-6**.  In order to fuzzify the inputs, the following membership functions are chosen namely Low, Medium and High.

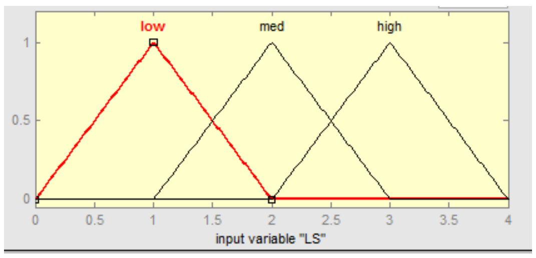


**Figure-4 Fuzzification of Average life span**

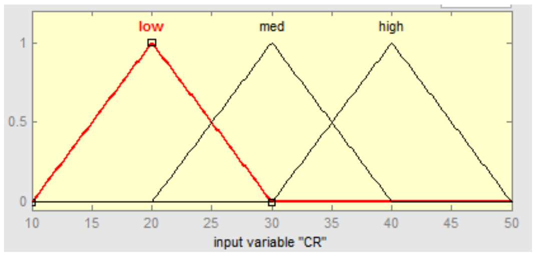


**Figure-5 Fuzzification of Comment Ratio**

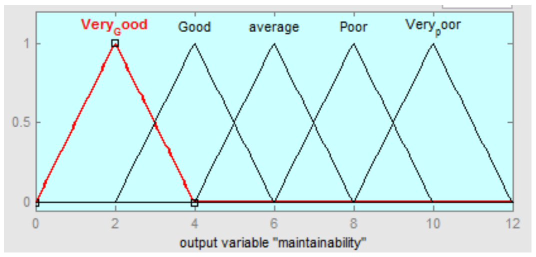


**Figure-6 Fuzzification of output variable – maintainability**

The toolbox includes 11 built-in membership function types. These 11 functions are, in turn, built from several basic functions:
Piecewise linear functions
The Gaussian distribution function
The sigmoid curve
Quadratic and cubic polynomial curves
The simplest membership functions are formed using straight lines. Of these, the simplest is the triangular membership function, and it has function name trimf. This function is nothing more than a collection of three points forming a triangle. The trapezoidal membership function, trapmf has a flat top and really is just a truncated triangle curve. These straight line membership functions have the advantage of simplicity.

## Rule Base

In Rule Base, we have to define rules. When the output is calculated, these rules are exercised. In first step, all the linguistic sets have been defined for both input and output. All possible combination of inputs is considered which leads to 3^4 i.e. 81 sets. The maintainability in case of all eighty-one combinations is classified as either Very Good, Good, Average, Poor or Very Poor by expert opinion. These lead to formation of 81 rules for the fuzzy model. All of them are shown as below:

1. If (LS is low) and (CR is low) then maintainability is very good.

2. If (LS is med) and (CR is med) then maintainability is average.

3. If (LS is high) and (CR is high) then maintainability is very poor.

4. If (LS is low) and (CR is med) then maintainability is good.

5. If (LS is med) and (CR is low) then maintainability is very good.

6. If (LS is med) and (CR is high) then maintainability is poor.

7. If (LS is high) and (CR is med) then maintainability is poor.

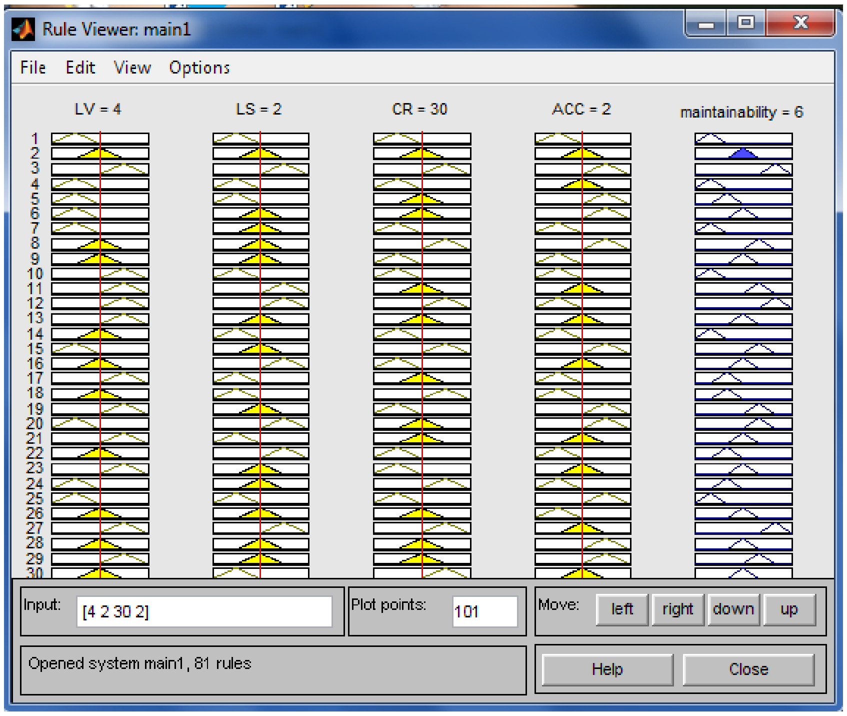
8. If (LS is high) and (CR is low) then maintainability is average.

9. If (LS is low) and (CR is high) then maintainability is good.

After the inputs are fuzzified, we know the degree to which each part of the antecedent is satisfied for each rule. If the antecedent of a given rule has more than one part, the fuzzy operator is applied to obtain one number that represents the result of the antecedent for that rule. This number is then applied to the output function. The input to the fuzzy operator is two or more membership values from fuzzified input variables. The output is a single truth value.

**Defuzzification**

The input for the defuzzification process is a set and output is a single number. As much as fuzziness helps the rule evaluation during the intermediate steps, the final desired output for each variable is generally a single number. However, the aggregate of a fuzzy set encompasses a range of output values, and so must be defuzzified in order to resolve a single output value from the set. Perhaps the most popular defuzzification method is the centroid calculation, which returns the center of area under the curve. There are five built-in methods supported: centroid, bisector, middle of maximum, largest of maximum, and smallest of maximum.



Input: [4 2 30 2]

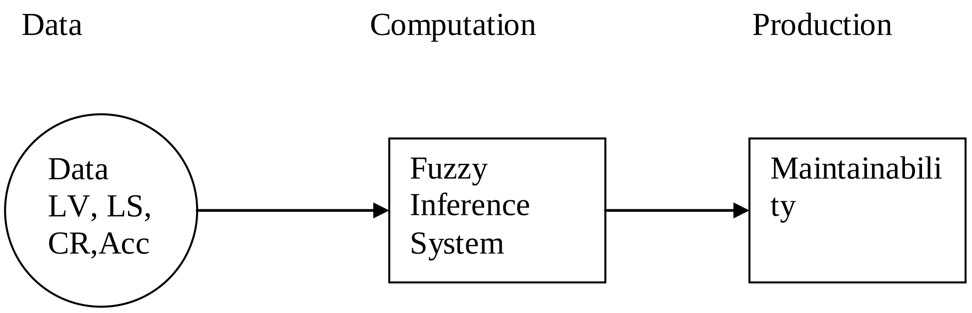
Plot points: 101

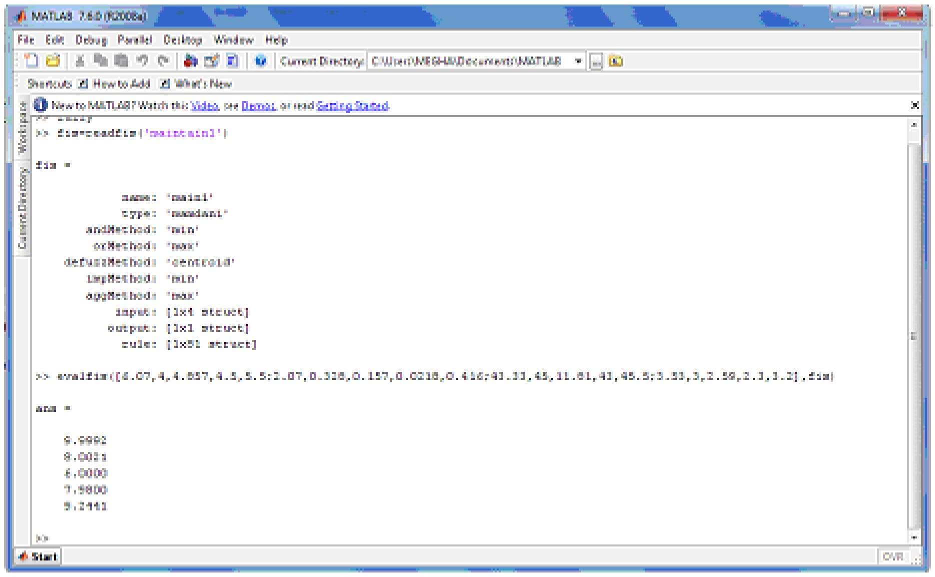
Opened system main1, 81 rules

Figure-7: Defuzzification of Output

## Experimental Study and Results

It is generally seen that maintainability very much depends on the type of data. We have tried to measure the maintainability of software. We have tried to show by arguments and by some empirical analysis that widely used complexity metrics like lines of code, Halstead's Software Science Metrics and others may not be appropriate to measure the complexity of the software. These metrics may not be appropriate to measure the maintenance cost and the efforts that are used to maintain software. In these we have considered the five software projects of undergraduate engineering students. After considered the projects, apply the different attributes on these projects, different values will be come from the different projects of all the four parameters. The input data is the value of the four parameters and the processing or computation takes place in the fuzzy inference system and the output will be the maintainability. The implementation has been done in MATLAB.




MATLAB 7.6.0 (R2008a)
File Edit Debug Parallel Desktop Window Help
Current Directory: C:\Users\MEGHA\Documents\MATLAB
Shortcuts How to Add What's New
New to MATLAB? Watch this Video, see Demos, or read Getting Started.
Start
OVR

```
>> fis=readfis('maintain1')

fis =

           name: 'maint1'
           type: 'mamdani'
      andMethod: 'min'
       orMethod: 'max'
   defuzzMethod: 'centroid'
      impMethod: 'min'
      aggMethod: 'max'
          input: [1x4 struct]
         output: [1x1 struct]
           rule: [1x51 struct]

>> evalfis([6.07,4,4.857,4.5,5.5;2.07,0.328,0.157,0.0218,0.416;43.33,45,11.81,43,45.5;3.53,3,2.59,2.3,3.2]',fis)

ans =

    9.9992
    8.0021
    6.0000
    7.9800
    9.2441

>>
```

**Figure-8: Snapshot of Matlab**

## Experimental Result

In order to validate the model, we have considered five software projects of undergraduate engineering students. They were chosen only when proper set of input Variables were available. The maintainability was also calculated using the proposed fuzzy model. The results are shown in **Table-1**.

| 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 |

**Table-1: Value of maintainability**

In these graphs all the four parameters were shown on one axis and the project no were shown on other axis. It can be seen there is hardly any co-relation between four parameters and the maintainability. These four parameters cannot individually predict the maintainability. Thus the fuzzy model is validated and that the integrated value of the maintainability gives the better result than any individual input metric is also verified with the help of results as shown in above Table.

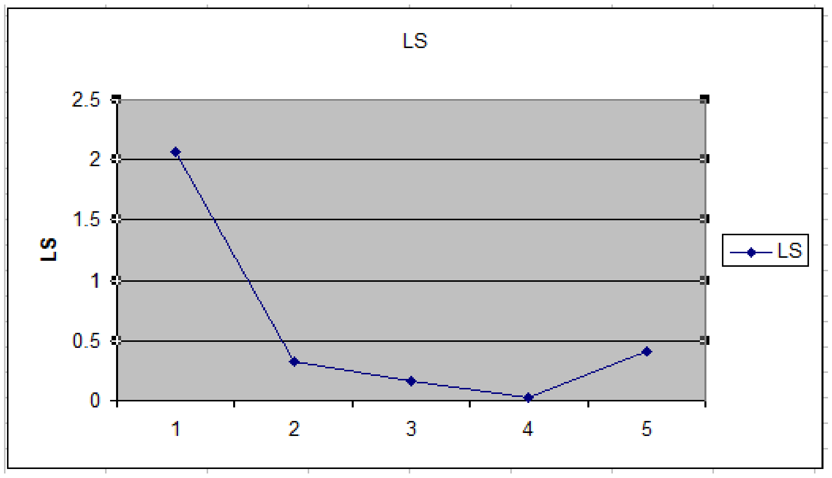


**Figure-9: Graph of Live Span**

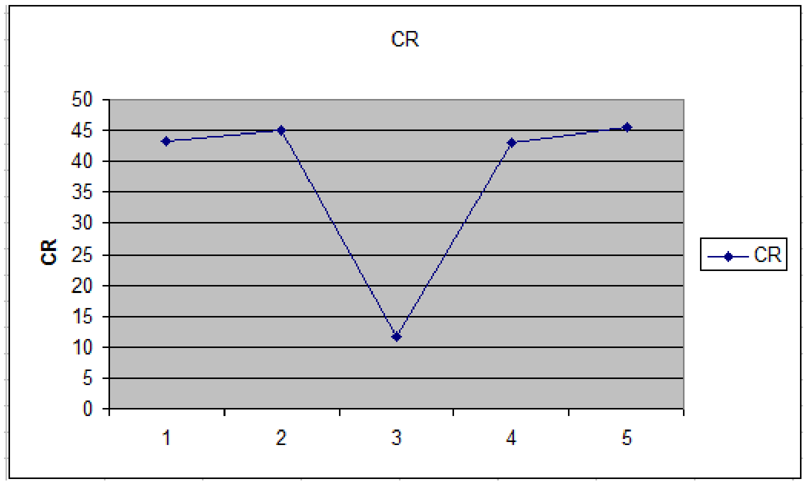


**Figure-10: Graph of Comment Ratio**

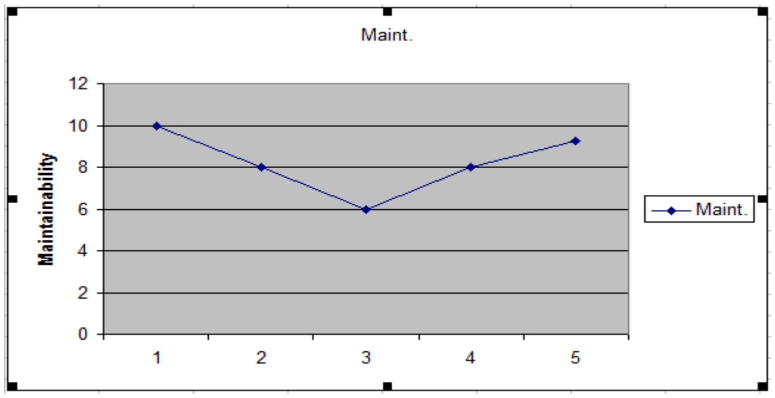


**Figure-11: Graph of Maintainability**

## Conclusion

This experimental provides details of experimental setup means considered data and full arrangements of experiment. It describes data that is various factors. Experimental Results section describes the outcome of experiment. It includes the value of maintainability of different projects in a tabular form corresponds to the different factors. The evaluation of maintainability using an excel graph. Some traditional metrics are there to measure the maintainability of software metrics like- line of code, Halstead's Software Science Metrics and Mc Cabe's Cyclomatic Complexity. But these metrics are not suitable to measure the maintainability of the software. It is generally seen that maintainability very much depends on the type of Programs. We have tried to show by arguments and by some empirical analysis that widely used traditional metrics may not be appropriate to measure the complexity of the software.

The time spent and efforts required for keeping software operational consumes about 40-70% of cost of the entire life cycle This study proposes a four parameters that integrated to measure the software maintainability. This study will evaluate how to reduce the maintenance cost and the efforts by using these parameters. So, we have developed fuzzy based model for measuring the software maintainability.

## Contribution of Present Work

A lot of techniques developed that included a number of factors to measure the maintainability like- Method Coupling (MC), Average Method Coupling (AMC), Class Coupling (CC). Aggarwal presented a model that takes a number of factors as input and measure maintainability. We are about on four factors like Average number of live variable, Average live Span, Comment Ratio, and Average Cyclomatic Complexity to measure the maintainability. We realized that these factors will provide more accurate view of maintainability for software. Maintainability can be estimated with the help of fuzzy model and the results prove that the integrated value of the maintainability gives the better results than any individual input metric is also verified with the help of empirical results. As we studied the different value of the four parameters, the value of these parameters should be low so that the maintainability will be low and the maintenance cost will be reduced. The efforts required for measuring the maintainability will be reduced. Minimizes or eliminates costly downtime increases profitable uptime. Fuzzy model helps a lot to validate these attributes.

## Future Work

Future work that can be done in this field to improve the accuracy of measurement, so as such system can be developed. We suggest the validation of this model in real time. When some error will be introduced in the project the time taken for correction of this error in maintenance time will be calculated.

## References

1- K.K. Aggarwal. 'Measurement of Software Maintainability Using a Fuzzy Model' Journal of Computer Sciences 1(4):538-542, 2005

2- Priyanka Dhankhar, Harish Mittal 'Software Maintainability in Object Oriented Software' in 2010 Proceedings conference 8th may 2010.

3- Alain April. "Software Maintenance Maturity Model: The software Maintenance process model" 2004

4- Megha and Harish Mittal "Review of Software Maintainability Prediction Modeling" proceeding of 1st National Conference on Advances Computational Intelligence NCACI-2011 9th July 2011

5- Megha and Harish Mittal "Fuzzy Based Software Maintainability Prediction Modeling" proceeding of 1st National Conference on Advances Computational Intelligence NCACI-2011 9th July 2011