IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661, p- ISSN: 2278-8727Volume 11, Issue 6 (May. - Jun. 2013), PP 01-14
www.iosrjournals.org

A Combined Approach of Software Metrics and Software Fault Analysis to Estimate Software Reliability

[1]Indu Sharma, [2]Ms. ParveenBano
[1]*Computer Science and Engineering Deptt, PDM College of Engineering, Bahadurgarh, Haryana (India)*
[2]*Asst. Prof, PDM College of Engineering, Bahadurgarh, Haryana (India)*

Abstract: *This paper presents a fault prediction model using reliability relevant software metrics and fuzzy inference system. For this a new approach is discussed to develop fuzzy profile of software metrics which are more relevant for software fault prediction. The proposed model predicts the fault density at the end of each phase of software development using relevant software metrics. On the basis of fault density at the end of testing phase, total number of faults in the software is predicted. The model seems to useful for both software engineer as well as project manager to optimally allocate resources and achieve more reliable software within the time and cost constraints. To validate the prediction accuracy, the model results are validated using PROMISE Software Engineering Repository Data set.*

Keywords: *Reliability Relevant Software Metrics, Software Fault Prediction, Fault Density, Fuzzy profile, Fuzzy Inference System (FIS)*

I. Introduction

Software has become so essential to human in their daily lives that today it is difficult to imagine living without devices controlled by software. Software reliability and quality has become the primary concern during the software development. It is difficult to produce fault-free software due to the problem complexity, complexity of human behaviors, and the resource constrains. Failure is an unavoidable phenomenonin all technological products and systems. System failures due to the software failure are very common and results undesirablesconsequences which can adversely affect both reliability and safety ofthe system. Now a day's most of the software development activity is performed in labor-intensive way. This may introduce various faults across the development, causing failures in near future. Therefore, there is a growing need ensure the reliability of these software systems by fixing the faults as early as possible.

Moreover, it is well known that earlier an error is identified, the better and more cost effectively it can be fixed. Therefore, there is a need to predict these software faults across the stages of software development process. IEEE defines software reliability as "the probability of a software system or component to perform its intended function under the specified operating conditions over the specified period of time". In other words it can also be defined as "the probability of failure-free software operation for a specified period of time in a specified environment". Software reliability is generally accepted as the key factor of software quality since it quantifies software failures that make the system inoperative or risky. A software failure is defined as "the departure of external result of program operation from requirements", whereas a fault is defined as "the defect in the program that, when executed under particular conditions, causes a failure". To further elaborate, a software fault is a defective, missing, or extra instruction or set of related instructions that may cause of one or more actual or potential failures when executed. Software reliability has roots in each step of the software development process and can be improved by inspection and review of these steps. Generally, the faults are introduced in each phase of software life cycle and keep on propagating to the subsequent phases unless they are detected through testing or review process. Finally, undetected and/or uncorrected faults aredelivered with software, causing failures. In order to achieve high software reliability, the number of faults in delivered code should be at minimum level. Software reliability can be estimated or predictedthrough various software reliability models. These models uses failure datacollected during testingand/or field operation to estimate or predict the reliability. This becomes late and sometimes infeasible for taking corrective actions. One solution to this problem may be predicting the faults across the stages of development process so that appropriate actions can be taken to mitigate or prevent these faults. Moreover, since the failure data are not available during the early phases of software life cycle, we have to dependent on the information such as reliability relevant software metrics (RRSM), expert opinions and similar or earlier project data. This paper proposes a multistage fault prediction model using reliability relevant software metrics (RRSM) that predicts the total number of faults at the end of each phase of software life cycle.

Software Reliability includes the following Parameters:-

Software measurement

- Software measurement is necessary for achieving the basic management objectives of prediction, progress, and process improvement.

Software Metrics

- Metrics which are used to evaluate the software processes, products and services is referred to as the software metrics.

Software Defect

- Defect (or Fault or bug) is a result of an entry of erroneous information into software. "Defects are raised when expected and actual test results differ".

1.1 Key Issues in Software Reliability

The nature of software engineering requires measurements to be made, as more rigorous methods for the purpose of production planning, monitoring, and control are needed, else the level of risk of the software projects may become immense, and there are chances that the software production gets out of industrial control. This will result in damage to both software producers (e.g., higher costs, schedule slippage) and users (e.g., poor quality products, late product delivery, and high prices).It should be effective and in order to make proper use of the resources devoted to it, software measurement should deal with important development issues, i.e., measurement should be done within industrial interest. In this regard, software measurement may serve several purposes, but it depends on the knowledge about a process of product.

1.2 Problem Definition

- As the software development begins there also exists the probability of occurrence of some defect in the software system.
- Software defects plays important role to take the decision about when the testing will be stopped.
- Software defects are one of the major factors that can decide the time of software delivery.
- To perform these kind of analysis we need to identify some of the important code metrics and the system metrics. In this work, we have to integrate the software metrics analysis along with fault analysis to identify the reliability of a software system. The presented work will be performed for any module based system.

II. Related Work

A lot of efforts have been made for software reliability prediction and assessment using various models . Gaffney and Davis of the Software Productivity Consortium developed a phase-based model, which makes the use of fault statistics obtained during the technical review of requirements, design, and the coding to predict reliability.

One of the earliest and well known efforts to predict software reliability during the early phase was the work initiated by the Air Force's Rome Laboratory. They developed a prediction model of fault density that can be transformed into failure rates. To do this the researchers selected a number of factors that they felt could be related to fault density at the earlier phases.

In a similar study, Agresti and Evancopresented a model to predict defect density based on the product and process characteristics for Ada program. Moreover, there are many papers advocating statistical models and software metrics. Most of them are based on size and complexity metrics.

A study was conducted by Zhang and Pham to find the factors affecting software reliability. The study found thirty two potential factors involved in various stages of the software life cycle. In another study conducted by Li and Smidt, thirty reliability relevant software engineering measures have been identified. They have developed a set of ranking criteria and their levels for various reliability relevant software metrics, present in the first four phases of software life cycle.

Kumar and Mishra made an effort for early software reliability prediction considering the six top ranked measures given by and software operational profile. Sometimes, it may happen that some of these top ranked measures are not available, making the prediction unrealistic. Also they have considered only product metrics and ignored process metrics that influence software reliability.

Recently, Pandey and Goyalhave developed an early fault prediction model using process maturity and software metrics. In their model they have developed the fuzzy profiles of various metrics in different scale and have not explained the criteria used for developing these fuzzy profiles.

Following are the general observations from the literature review:

- Software reliability is a function of number of faults present in the software.
- Software metrics plays a vital role in fault prediction in the absence of failure data.
- Early phase software metrics are of fuzzy nature.
- Software metrics follow either linear or logarithmic nature.

III. Approaches Used

A number of different approaches have been used to detect and remove faults in Software. We will use three major approaches to detect and remove the Defects and faults in software. One isSoftware Defects and Object Oriented approach, Second is Musa's and Monte Carlo Simulation
Approach and new is Fuzzy approach.

3.1 Software Defects and Object Oriented approach

Object oriented design, today, is becoming more popular in
Software development environment and Object Oriented design metrics is an essential part of software environment. The main objective of analyzing these metrics is to improve the quality of the software. Detection and removal of defects prior to the customer delivery is extremely important in the software development**.** In this paper first we have discussed the Chidamber and Kemerer metrics suite first, then the methodology & analysis to answer above mentioned questions [1998].

CK Metrics Suite [2003-2009]

Weighted Method per Class (WMC) – WMC measures the complexity of a class. WMC is a predictor of how much time and effort is required to develop and maintain the class. A large number of methods also mean a greater potential impact on derived classes, since the derived classes inherit the methods of the base class. If we analyze the WMC then we will find that high WMC leads to more faults which increases the density of bugs and decreases quality [2009].

Depth of inheritance Tree (DIT) – DIT metric is the length of the maximum path from the node to the root of the
tree. The deeper a class is in the hierarchy, the more methods and variables it is likely to inherit, making it more complex. High depth of the tree indicates greater design complexity. Thus it can be hard to understand a system with many inheritance layers [2009].

Number of Children (NOC) – It is equal to the number of immediate child classes derived from a base class. NOC measures the breadth of a class hierarchy. A high NOC indicates several things like- High reuse of base class, base class may require more testing, improper abstraction of parent class etc. [2009]

Coupling Between Objects (CBO) - Two classes are coupled when methods declared in one class use methods or
Instance variables defined by the other classes. Multiple accesses to the same class are counted as one access. Only method calls and variable references are counted. An increase of CBO indicates the reusability of a class will decrease; also a high coupling has been found to indicate fault proneness. Thus, the CBO values for each class should be kept as low as possible [2009].

3.2.1 Musa's Basic Execution Time Model

The Musa's Basic Execution Time Model [2004] is an example of a prediction model. Prediction models are used to make software reliability predictions early in the development phase that help make software engineering decisions in the design stage.

Musa's Basic Execution Time Model was one of the first software reliability models used to model reliability based on the execution time of software components instead of the calendar time components are in use.
Musa's Basic Execution Time Model calculates the reliability of a software system using the Poisson distribution (Figure 4, 5). The data required to implement Musa's Basic Execution Time Model include the time elapsed between software failure and the actual calendar time the software was in use.

$$p(x,\lambda) = \frac{e^{-\lambda}\lambda^{x}}{x!}$$

Where x = time and λ = number of failures

Figure 4: Poisson distribution Probability

$$F(x,\lambda) = \sum_{i=0}^{x} \frac{e^{-\lambda}\lambda^{i}}{i!}$$

Where x, i = time and λ = number of failures

Figure 5: Poisson distribution Cumulative Distribution

Musa's Basic Execution Time Model [2004] is limited by the assumption that new faults are not introduced after the correction of old faults, and the assumption that the number of faults decreases over time. These assumptions limit the model, because as any experienced programmer knows, making fixes to code does not guarantee the updated code will be bug free.

3.2.2 Monte Carlo Simulation

The Monte Carlo Simulation was developed in the 1940s as part of the atomic bomb program. Monte Carlo Simulations are used to model many different processes including processes for physics, finance and system reliability. Monte Carlo Simulations [2005] are generally used for complex problems that are either difficult to solve or no definite solution exists. Monte Carlo Simulations are generally referred to as "brute force" methods [2005] for calculating reliability which means these simulations are more expensive both money wise and time wise than other reliability processes.

3.3 Fuzzy Approach

In our proposed work we are presenting a Fuzzy approach to analyze the software defects and software metrics. The complete work is divided in 3 main steps.

1. Analysis the software modules under the complexity constraints for different modules.
2. Define a classifier to analyze the software modules under the software fault and fault criticality parameters.
3. Relate the software module complexity and the software faults to analyze the quality of the software system.

IV. Proposed Model

Prediction of faults is desirable for any industry and attracts both engineers as well as managements. For software industry it provides an opportunity for the early identification of software quality, cost overrun and optimal development strategies. During earlier phases of software development; predicting the number of faults can reduce the efforts for additional reviews and more extensive testing [7]. The model architecture is shown in Figure1. Stages present in the proposed structure are similar to the waterfall model, a well-known software development process model. It divides the structure into four consecutive phase I, II, III, and IV i.e. requirement, design, coding, and testing phase respectively. Phase-I predicts the fault density at the end of requirement phase using relevant requirement metrics. Phase-II predicts the fault density at the end of design phase usingdesign metrics as well as output of the requirements phase. Similarly at phase-III besides the using coding metrics, output of phase-II is also considered as input to predict the fault density at the end of coding phase. Finally, the phase-IV predicts the fault density using testing metrics as well as output of the coding phase. The proposed model considers three requirements metrics (RM): a) requirements complexity (RC), b) requirements stability (RS), and c) review, inspection and walk-through (RIW) as input. Similarly, at design phase two design metrics (DM): a) design team experience (DTE) and b) process maturity (PM) are considered as input. Two coding phase metrics (CM): a) coding team experience (CTE) and b) defined process followed (DPF) are taken as input. Finally, testing phase metrics (TM): a) testing team experience (TTE), b) stake-holders involvement (SI) and c) size of the software (KLOC) are taken as input. The outputs of the model are fault density indicator at the end of requirements phase (FDR), design phase (FDD), coding phase (FDC) and testing phase (FDT). It is important to mention here that these metrics may fallow either linear or logarithmic scale based on their nature.

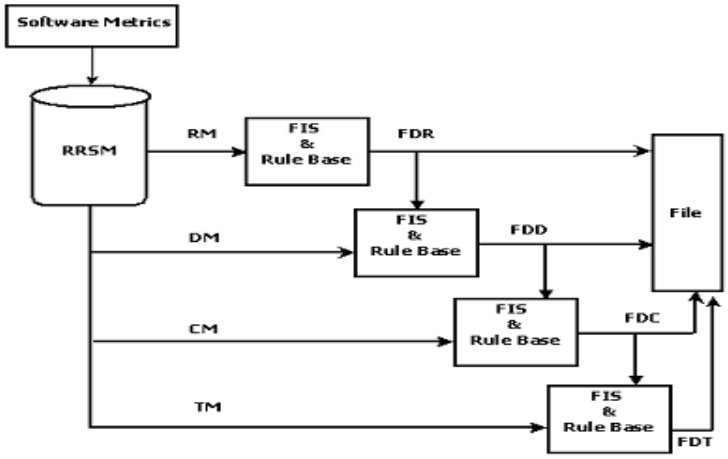

Figure 1. Software fault prediction model

Table 1. Phase-wise input/output variable

Phase	Input Variables	Output Variables
Requirement	RC, RS, RIW	FDR
Design	FDR, DTE, PM	FDD
Coding	FDD, CTE, DPF	FDC
Testing	FDT, TTE, SI, KLOC	FDT

V. Research Design

- Define Number of Faults Respective to Each Test Case
- Generate the Test Sequence for the Software Project
- Define Fault Parameters that can influence the Test and Reliability
- Define fault Categories and Subcategories
- Assign the weightage to each Sub Category
- Define no. of faults respective to each test case
- Define the Fuzzy Rule based to Software Faults
- Estimate the Software Risk based on defined rule set and Define the Fuzzy Rule based to Software Faults
- Analysis of Results

5. 1 MATLAB

MATLAB is a package that has been purpose-designed to make computations easy, fast and reliable. It is installed on machines run by Bath University Computing Services (BUCS), which can be accessed in the BUCS PC Labs such as those in 1 East 3.9, 1 West 2.25 or 3 East 3.1, as well as from any of the PCs in the Library. The machines which you will use for running MATLAB are SUN computers that run on the UNIX operating system. (If you are a PC or Mac fan, note that this is a quite different environment from what you are used to. However you will need only a small number of UNIX commands when you are working with MATLAB.

MATLAB started life in the 1970s as a user-friendly interface to certain clever but complicated programs for solving large systems of equations.

Basic MATLAB is good for the following.

• Computations, including linear algebra, data analysis, signal processing, polynomials and interpolation, numerical integration (quadrature), and numerical solution of differential equations.

• Graphics, in 2-D and 3-D, including colour, lighting, and animation.

Features

- Matlab is an interactive system for doing numerical computations.
- A numerical analyst called Cleve Moller wrote the first version of Matlab in the 1970s. It has since evolved into a successful commercial software package.
- Matlab relieves you of a lot of the mundane tasks associated with solving problems numerically. This allows you to spend more time thinking, and encourages you to experiment.

- Matlab makes use of highly respected algorithms and hence you can be confident about your results.
- Powerful operations can be performed using just one or two commands.
- You can build up your own set of functions for a particular application.

5.2 Calculation of different Risk Factors

5.2.1 Low Risk (Factor)

```
Function factor1=LowRisk (Factor)
      a = .3;
      b = .5;
if (Factor < a)
            factor1=1;
elseif (Factor >=a && Factor < b)
                                    factor1=((b - Factor)) / (b-a);
else
                factor1=0;
end
```

5.2.2 High Risk (Factor)

```
function  factor=HighRisk  (Factor)
  a = .5;
  b = .8;
  c=1;
if (Factor ==b)
factor=1;
elseif (Factor >= a && Factor < b)
factor=( Factor-a) / (b-a);
elseif (Factor>= b && Factor<= c)
factor=(c- Factor) / (c-b);
else
factor=0;
end
```

5.2.3 Medium Risk (Factor)

```
function  factor=MediumRisk  (Factor)
        a = .2;
        b = .5;
        c=.8;
if (Factor == b)
factor=1;
elseif (Factor >= a && Factor < b)
factor= (Factor-a) / (b-a);
elseif (Factor>= b && Factor< c)
factor=(c- Factor) / (c-b);
else
factor=0
end
```

5.3 Relation and Test Cases between Modules

Relation
Test case
Operations
Attributes
Nested Components

5.4 Implementation

The model is based on the fuzzy logic and implemented in MATLAB. The basic steps of the model are identification of RRSMs as input/output variables, development of fuzzy profile of these input/output variables defining relationship between input and output variables and fault prediction at the end of software life cycle using fuzzy inference system (FIS). These basic steps can be grouped into three broad phases as follows: a) information gathering phase, b) information processing phase, and c) fault prediction phase.

5.4.1 Information gathering phase

The quality of fuzzy approximation depends mainly on the quality of information collected and expert opinion. The information gathering phase is often considered the most vital step in developing a fuzzy inference system and includes the following steps:

5.4.1.1 Selection of input / output variables

A list of RRSMs applicable to various phase of software life cycle is given in Appendix A. Ten input and four output variables are identified, as shown in Table 1, for the purpose of predicting number of faults in the software. Input variables are the RRSMs relevant to each phase of software life cycle and output variables are the fault densities at the end of each phase. We found that metrics RC, RS, and RIW are more suitable for requirement phase and influence to the requirements faults. If RC is more, the number of faults will be more, but this is not true for RS. Similarly, if there are more reviews andinspection, more faults will be detected and corrected leaving fewer faults in the requirements phase. For design phase, two metrics DTE and PM are considered because both of these metrics are responsible for error free softwaredesign. For higher value of DTEand PM, there will be lower number of design faults inthe software. At coding phase, CTE and DPF metrics is found more suitable to affect the coding faults. In general it is found that if CTE and DPF are more, the number of faults will be less. Lastly, for testing phase, three metrics TTE, SI and KLOC are taken which can influence the fault density at this phase.

5.4.2 Fuzzy profile development

Input /output variables identified at the previous stage are fuzzy in nature and characterized by fuzzy numbers. Fuzzy numbers are a subset from the real numbers set, representing the uncertain values. All fuzzy numbers are related to degrees of membership, which state how true it is to say if something belongs or not to a determined set. There are various types of fuzzy numbers and its nomenclature is, in general, associated with its format, such as: sine numbers, bell shape, polygonal, trapezoids, triangular, and so on.

Triangular fuzzy numbers (TFN) are convenient to work with because in real space, adding two TFN involves adding the corresponding vertices used to define the TFNs. Similar simple formulas can be used to subtract or find the image of TFNs. Also TFNs are well suited to modeling and design because their arithmetic operators and functions are developed, which allow fast operation on equations. Because of these properties we have considered TFNs, for all input/output variables. We have divided input variables into five linguistic categories as: i.e. very low (VL), low (L), moderate (M), high (H) and very high (VH), and for output variables we have considered seven linguistic categories i.e. very very low (VVL), very low (VL), low (L), moderate (M), high (H), very high (VH), and very very High (VVH). The data, which may be useful for selecting appropriate linguistic variable, is generally available in one or more forms such as expert's opinion, software requirements, user's expectations, record of existing field data from previous release or similar system, etc. Fuzzy membership functions are generated utilizing the linguistic categories such as identified by a human expert to express his/her assessment and the nature of the metrics. As stated earlier, that software metrics may be either linear or logarithmic nature. On the basis of this information, fuzzy profiles (FP) of each software metrics are developed using the following formula,

(a) For logarithmic nature software metrics,

$$FP = \left[1 - \frac{\log_{10} 1{:}5}{\log_{10} 5} \right]$$

The profiles may take the following values:

VL (0; 0; 0.14), L (0; 0.14; 0.32), M (0.14; 0.32; 0.57), H (0.32; 0.57; 1.00), and VH (0.57; 1.00; 1.00)

(b) For linear nature software metrics,

$$FP = \left[\frac{(0:4)}{4} \right]$$

The profiles may take the following values: VL (0; 0; 0.14), L (0; 0.14; 0.32), M (0.14; 0.32; 0.57), H (0.32; 0.57; 1.00), and VH (0.57; 1.00; 1.00)

Outputs are considered on logarithmic scale, and divided in seven linguistic categories as: Fuzzy profile range = [1-{log10 (1:7)} / {[log10 (7)}]; the profiles may take the following values: VVL (0; 0; 0.08), VL (0; 0.08; 0.17), L (0.08; 0.17; 0.29), M (0.14; 0.32; 0.57), H (0.17; 0.29; 0.44),
VH (0.44; 0.64; 1.00), VVH (0.64; 1.00; 1.00)

It is assumed that out of these ten input variables, only three variables (RIW, PM and DPF) follow linear nature and remaining variables follow logarithmic nature. All output variables are assumed to be following logarithmic nature. Fig. 2 to Fig. 15 shows membership functions and fuzzy profiles of all the selected input/output variables for visualization purpose.

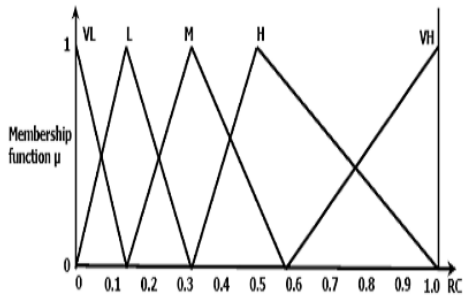

Figure 2. Fuzzy profile of RC

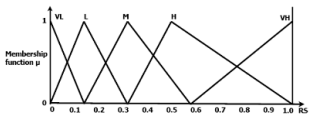

Figure 3. Fuzzy profile of RS

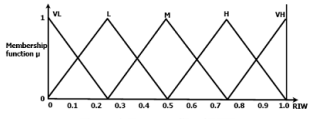

Figure 4. Fuzzy profile of RIW

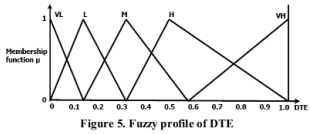

Figure 5. Fuzzy profile of DTE

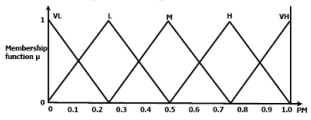

Figure 6. Fuzzy profile of PM

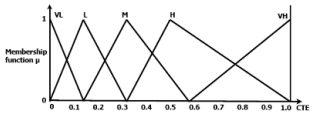

Figure 7. Fuzzy profile of CTE

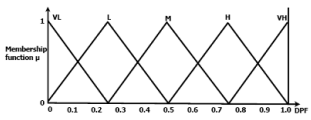

Figure 8. Fuzzy profile of DPF

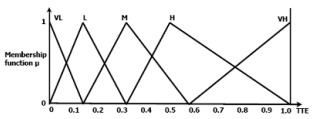

Figure 9. Fuzzy profile of TTE

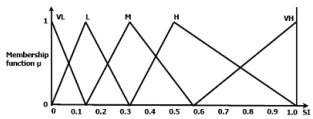

Figure 10. Fuzzy profile of SI

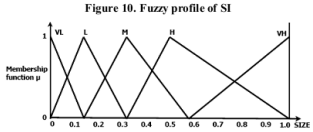

Figure 11. Fuzzy profile of SIZE

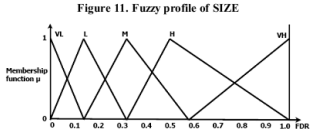

Figure 12. Fuzzy profile of FDR

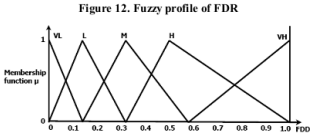

Figure 13. Fuzzy profile of FDD

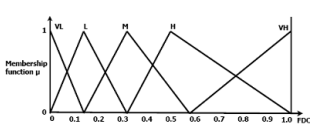

Figure 14. Fuzzy profile of FDC

5.4.3 Development of fuzzy rule base

The most important part of any inference system is the rules, and how they interact with each other to generate results. In general, rules are formed using various domain experts so that the system can emulate the inference of an actual expert. To develop fuzzy rule base, we can acquire knowledge from different sources such as domain experts, historical data analysis of similar or earlier system, and engineering knowledge from existing literature's. In our experiments, we generated some rules from the software engineering point of view and some from project

management viewpoints. All the rules take the form of `If A then B'. Table 2 to Table 5 show the fuzzy if-then rules required for each phase of software life cycle.

Table 2. Fuzzy rules at requirements phase

Rule	RC	RS	RIW	FDR
1	L	L	L	VL
2	L	L	M	L
3	L	L	H	M
.	.	.	.	.
.	.	.	.	.

Table 3. Fuzzy rules at design phase

Rule	FRP	DTE	PM	FDD
1	VL	L	L	VL
2	VL	L	M	VL
3	VL	L	H	L
.	.	.	.	.

Table 4. Fuzzy rules at coding phase

Rule	FDP	CTE	DPF	FDC
1	VL	L	L	VL
2	VL	L	M	VL
3	VL	L	H	L
.	.	.	.	.
.	.	.	.	.

5.4.4 Information Processing Phase

In this phase, the fuzzy system maps all inputs on to an output. This process of mapping is known as fuzzy inference process or fuzzy reasoning. Basis for this mapping is the number of fuzzy IF-THEN rules, each of which describes the local behavior of the mapping. The Mamdani fuzzy inference system is considered here for all the information processing. We have taken the centroid value of each fuzzy profile for computation purpose. Centroid value of each fuzzy profile can be computed from its fuzzy number and these are shown in the Table 6. Another vital part of information processing is the defuzzification process, which derives a crisp value from a
number of fuzzy values. Various defuzzification techniques are Centroid, Bisector, Middle of maximum, Largest of maximum, Smallest of maximum, etc. The most commonly used method is the centroid method,which returns the centre of area under the curve, is considered here for defuzzification.

5.4.5 Fault Prediction Phase

On the basis of fault density at the end of testing phase, total number of faults in the software is predicted. The faults are predicted from FTP value, which is an indicator offault density at the end of testing phase. The number of faults detected is proportional to the both size of the software and the fault density indicator value and can be found as:

FaultP= c1* LOC* FTP

However, fault detection process is not exactly linear with size. As size of a software increases, portion of faults detected decreases due to saturation, time and efforts requirements, increased possibility due to interactions among the variables and instructions. Therefore, the FTP value is divided by (1+exp (- LOCi /C2)), where the value of C2 scales the effect of LOC value. Thus (1) becomes,

FaultPi=C1*LOCi*FTPi/(1+exp(-LOCi/C2))

Where FaultPi, is the total number of predicted faults in the ith software project, LOC is the size of ith project, FTPi, is the fault density indicator at the end of testing phase of project i, and C1 and C2 are two constants obtained through learning. Value of C1 and C2 are obtained from available project data. The values of C1 and C2 for current project are obtained as 0.04 and 107 respectively. The proposed fuzzy inference model is generic in nature to capture variation in faults present in the software.

MatLabImplementation

```
clear all;
close all;
clc;
close all;
clear all;
aa='attributes.csv';
fid = fopen(aa);
   data1x=textscan(fid,'%s %s %s','delimiter',',');
fclose(fid);
componentid=data1x{1,1};
attributeid=data1x{1,2};
weightage=data1x{1,3};
componentid=str2mat(componentid(2:length(componentid),1));
attributeid=str2mat(attributeid(2:length(attributeid),1));
weightage=str2mat(weightage(2:length(weightage),1));

componentid=str2num(componentid);
attributeid=str2num(attributeid);
weightage=str2num(weightage);
ucomponentid=unique(componentid);
numberofcomponents=length(ucomponentid);
sprintf('Number of components %d',numberofcomponents)
fori=1:numberofcomponents
CMPXAtt(i)=0;
for j=1:length(componentid)
ifcomponentid(j)==ucomponentid(i)
```

```
CMPXAtt(i)=CMPXAtt(i)+   attributeid(j)* weightage(j);
end
end
end
display('Complexity Attributes of all Components');
CMPXAtt
aa='operations.csv';
fid = fopen(aa);
   data1x=textscan(fid,'%s %s %s','delimiter',',');
fclose(fid);

componentid=data1x{1,1};
operation=data1x{1,2};
weightage=data1x{1,3};
componentid=str2mat(componentid(2:length(componentid),1));
operation=str2mat(operation(2:length(operation),1));
weightage=str2mat(weightage(2:length(weightage),1))
componentid=str2num(componentid);
operation=str2num(operation);
weightage=str2num(weightage);

ucomponentid=unique(componentid);
numberofcomponents=length(ucomponentid);

fori=1:numberofcomponents
CMPXOp(i)=0;
for j=1:length(componentid)
ifcomponentid(j)==ucomponentid(i)
CMPXOp(i)=CMPXOp(i)+   operation(j)* weightage(j);
end
end
end
display('Complexity Operations of all Components');
CMPXOp
aa='nestedcomponents.csv';

fid = fopen(aa);
   data1x=textscan(fid,'%s %s %s','delimiter',',');
fclose(fid);

componentid=data1x{1,1};
nestedcomponents=data1x{1,2};
weightage=data1x{1,3};
componentid=str2mat(componentid(2:length(componentid),1));
nestedcomponents=str2mat(nestedcomponents(2:length(nestedcomponents),1));
weightage=str2mat(weightage(2:length(weightage),1));
componentid=str2num(componentid);
nestedcomponents=str2num(nestedcomponents);
weightage=str2num(weightage);
ucomponentid=unique(componentid);
numberofcomponents=length(ucomponentid);

fori=1:numberofcomponents
CMPXNC(i)=0;
for j=1:length(componentid)
ifcomponentid(j)==ucomponentid(i)
CMPXNC(i)=CMPXNC(i)+   nestedcomponents(j)* weightage(j);

end
end
```

```
end
display('Complexity Nested Components of all Components');
CMPXNC

alpha=0.33;
beta=0.33;
gamma=0.33;

CMPXcc = alpha *sum(CMPXAtt)+beta *sum(CMPXOp)+gamma*sum(CMPXNC);
display('Code Complexity of the System');

CMPXcc
close all;
clear all;
%[filename pathname]=uigetfile('*.csv');
  %   aa = strcat(pathname,filename);
aa='testcase.csv';
fid = fopen(aa);
   data1x=textscan(fid,'%s %s','delimiter',',');
fclose(fid);
testcase=data1x{1,1};
fault=data1x{1,2};
testcase

testcase1=str2mat(testcase(2:length(testcase),1));
fault1=str2mat(fault(2:length(fault),1));

%dmos_std11=str2num(dmos_std1);
fault11=str2num(fault1);
testcase11=str2num(testcase1);
[r c]=size(fault11)
figure,bar(fault11)
utestcase=unique(testcase11);%%%%%%%

utestcase
ufault=unique(fault11);%%%%%%%
ufault

fori=1:length(utestcase)
   s=0;
for j=1:r
if testcase11(j)==utestcase(i)
        s=s+ fault11(j);
end
end
weight(i)=s/10;
end
weight
tweight=0;
fori=1:length(weight)
r1=LowRisk(weight(i));
r2=MediumRisk(weight(i));
r3=HighRisk(weight(i));
fprintf('\nTestCase %d    %d  %d    %d   %d',i,weight(i),r1,r2,r3);
end
```

VI. Related Work

In Year 2010, Thomas J. Ostrand performed a work," Programmer-based Fault Prediction". Background: Author investigates whether files in a large system that are modified by an individual developer consistently contain either more or fewer faults than the average of all files in the system. The goal of the

investigation is to determine whether information about which particular developer modified a file is able to improve defect predictions.

In Year 2010, Ajeet Kumar Pandey performed a work," Fault Prediction Model by Fuzzy Profile Development of Reliability Relevant Software Metrics". This paper presents a fault prediction model using reliability relevant software metrics and fuzzy inference system. For this a new approach is discussed to develop fuzzy profile of software metrics which are more relevant for software fault prediction. The proposed model predicts the fault density at the end of each phase of software development using relevant software metrics.

In Year 2010, Dr. B.R. Sastry et al. tried to implement software metrics with aid of GUI & also analyzed relationships of metrics to determine quality and quantity of software attributes measured with regard of object oriented software development life cycle.

In Year 2010, Amjan Shaik et al. have done statistical analysis for object oriented software metrics on CK metric suite by validating the data collected from the projects of some students. Metrics data provided quick feedback for software designers and managers. They found out that if appropriately used; it could lead to a significant reduction in cost of the overall implementation and improvement in quality of the final product.

In Year 2009, C. Neelamegan et al. did survey on four object oriented quality metrics. In this study, the metrics for object oriented design focused on measurements that were applied to the class and design characteristics.

In Year 2008, Yue Jiang performed a work," Comparing Design and Code Metrics for Software Quality Prediction". The goal of this paper is to compare the performance of predictive models which use design-level metrics with those that use code-level metrics and those that use both. Author analyzes thirteen datasets from NASA Metrics Data Program which offer design as well as code metrics.

In Year 2007, Dr Kadhim M. Breesam validated a set of metrics empirically that could be used to measure the quality of an object oriented design in terms of the using class inheritance.

In Year 2003, Ruminate Subramanyam et al. conducted an empirical analysis on subset of C & K metrics suite in determining software defects.

In 1998, IEEE "Defines three software reliability approaches. This paper is going to examine two approaches: Musa's Basic Execution Time Model, and Monte Carlo Simulation and discuss how each approach can be applied to achieving software reliability.

In Year 1996, Victor R. Basili et al. conducted empirical analysis on Object Oriented metrics. The aim of the paper was to access these metrics as predictors of fault prone classes.

VII. Conclusion and Future Work

This paper has developed a fault prediction model using early phase software metrics and fuzzy inference system. Fuzzy profiles of metrics were developed by a human expert to express his/her assessment and the nature of the metrics. For this, ten software metrics are taken from PROMISE data set and fault density at the end of each phase is predicted using fuzzy inference system. For software professionals, this model provides an insight towards software metrics and its impact on software fault during the development process. For software project managers, the model provides a methodology for allocating the resources for developing reliable and cost- effective software. There are various models for fault prediction and each one has its own strengths, and weakness. An interesting open problem for the future work is to find the failure rate form these fault densities, and predict the software reliability, safety, availability from these values.

Appendix A: PROMISE database: Twenty six software project data

#	Project	RC F1	RS S7	RIW S3	DTE D1	PM D4	CTE D2	DPF D3	TTE T2	SI P5	Efforts E	Size K	Faults TD
1	1	M	L	VH	L	H	H	H	H	H	7108.82	6.02	148
2	2	L	H	VH	L	H	H	H	H	H	1308.08	0.90	31
3	3	H	H	VH	H	VH	VH	H	H	VH	18170.00	53.86	209
4	5	H	M	H	L	H	M	H	M	M	9434.00	14.00	373
5	7	L	M	VH	M	H	VH	H	M	VH	13888.27	21.00	204
6	8	M	H	H	H	M	H	M	M	H	8822.00	5.79	53
7	10	M	H	H	H	H	H	H	M	H	4410.00	4.84	29
8	11	H	H	H	H	H	H	H	H	H	14196.00	4.37	71
9	12	H	L	H	VH	H	M	M	H	H	13387.50	19.00	90
10	13	H	L	M	H	H	H	H	M	H	25449.60	49.10	129
11	14	VH	H	H	H	H	H	H	H	H	33472.00	58.30	672
12	15	H	VL	H	H	H	H	H	H	VH	34892.65	154.00	1768
13	16	L	M	H	H	H	H	H	H	VH	7121.00	26.67	109
14	17	L	M	M	M	H	M	H	L	M	13680.00	33.00	688
15	18	VH	VL	H	M	H	H	H	H	VH	32365.98	155.20	1906
16	19	H	M	H	H	H	H	H	M	H	12387.65	87.00	476
17	20	VH	VL	M	VL	H	VL	L	VL	H	52660.00	50.00	928
18	21	L	M	H	H	H	H	H	H	H	18748.00	22.00	196
19	22	M	L	M	H	H	M	L	M	H	28206.00	44.00	184
20	23	H	M	VH	L	H	H	H	H	H	53995.00	61.00	680
21	24	M	L	M	M	M	H	H	M	M	24895.00	99.00	1597
22	27	H	M	VH	M	H	L	M	M	M	14602.00	52.00	412
23	28	VH	L	VH	M	H	L	H	M	M	8581.00	36.00	881
24	29	M	VH	VH	VH	H	VH	H	VH	VH	3764.00	11.00	91
25	30	L	VH	VH	H	H	H	H	H	VH	1976.00	1.00	5
26	31	M	M	H	H	H	H	H	H	VH	15691.00	33.00	653

References

[1] ANSI/IEEE Standard Glossary of Software Engineering Terminology, IEEE STD-729, 1991.

[2] J. D. Musa, A. Iannino, K. Okumoto, Software Reliability: Measurement, Prediction, Application, McGraw-Hill Publishers, New York, 1987.

[3] C. Kaner, Software Engineering Metrics: What Do They Measure and How Do We Know, 10th International Software Metrics Symposium 2004.

[4] J. E. Gaffney, C. F. Davis, An Approach to Estimating Software Errors and Availability, Proceedings of 11th Minnow brook Workshop on Software Reliability 1988.

[5] J. E. Gaffney, J. Pietrolewiez, An Automated Model for Software Early Error Prediction(SWEEP), Proceedings of 13th Minnow brook Workshop on Software Reliability1990.

[6] Technical Report, Report Methodology for Software Reliability Prediction and Assessment, Rome Laboratory (RL) RL-TR-92-52, vol. 1 & 2, 1992

[7] W. W. Agresti, W. M. Evanco, Projecting Software Defects form Analyzing Ada Design, IEEE Transaction on Software Engineering, vol. 18, no. 11, pp. 988-997, 1992.

[8] T. J. Yu, V. Y. Shen, H. E. Dunsmore, (1988), An Analysis of Several Software Defect Models, IEEE Transaction on Software Engineering, vol. 14, no. 9, pp. 261-270, 1988.

[9] T. M. Khoshgoftaar, J. C. Munson, Predicting Software Development Errors Using Complexity Metrics, IEEE Journal on Selected Areas in Communication, vol. 8, no. 2, pp. 253-261, 1990.

[10] X. Zhang, H. Pham, An Analysis of Factors Affecting Software Reliability, The Journal of Systems and Software, vol. 50, no. 1, pp. 43-56, 2000.

[11] K. S. Kumar, R. B. Misra, An Enhanced Model for Early Software Reliability Prediction using Software Engineering Metrics, Proceedings of 2nd Int. Conf. on Secure System Integration and Reliability Improvement, pp. 177-178, 2008.

[12] A. K. Pandey, N. K. Goyal, A Fuzzy Model for Early Software Fault Prediction Using Process Maturity and Software Metrics, International Journal of Electronics Engineering, vol. 1, no. 2, pp. 239-245, 2009.

[13] M. S. Krishnan, M. I. Kellner, Measuring Process Consistency: Implications Reducing Software Defects, IEEE Transaction on Software Engineering, vol. 25, no. 6, pp. 800-815, 1999.

[14] D. E. Harter, M. S. Krishnan, S. A. Slaughter, Effects of Process Maturity on Quality, Cycle Time and Effort in Software Product Development, Management Science, vol. 46, pp. 451-466, 2000.

[15] K. S. Saravana, R. B. Misra, N. K. Goyal, Development of Fuzzy Software Operational Profile, International Journal of Reliability, Quality and Safety Engineering, vol. 15, no. 6, 581-597, 2008.

[16] T. Ross, Fuzzy Logic with Engineering Applications, Wiley-India, New Delhi 2005.

[17] L. A. Zadeh, Knowledge representation in fuzzy logic, IEEE Transactions on Knowledge and Data Engineering, vol. 1, pp. 89-100, 1989.

[18] M. Xie, G. Y. Hong, C. Wohlin, Software reliability prediction incorporating information from a similar project, The Journal of Systems and Software, vol. 49, pp. 43-48, 1999.

[19] J. B. Bowles, C. E. Pelaez, Application of fuzzy logic to reliability engineering, IEEE Proceedings, vol. 83, no. 3, pp. 435-449, 1995.

[20] E. H. Mamdani, Applications of fuzzy logic to approximate reasoning using linguistic synthesis, IEEE Transactions on Computers, vol. 26, no. 12, pp.1182- 1191, 1977.