Fuzzy Logic Driven Testability Measurement for an Object Oriented System

A Thesis

Submitted in fulfilment of the requirements for the award of the degree of

Master of Science

in

Mathematics and Computing

Submitted by

Harleen Kaur Ahuja

(Registration No. 301003009)

Under the supervision of

Dr. Rajesh Kumar

School of Mathematics and Computer Applications,

Thapar University,

Patiala-147004(PUNJAB)

INDIA

JULY 2012

Certificate

I hereby certify that the work is being presented in the thesis entitled "Fuzzy Logic Driven Testability Measurement for an Object Oriented System" in partial fulfilment of the requirements for the award of degree of Master of Science, School of Mathematics and Computer Applications, Thapar University, Patiala is an authentic record of my own work carried out under the supervision of Dr. Rajesh Kumar.

The matter presented in this thesis has not been submitted for the award of any other degree of this or any other university.

Harleen Kaur

(Harleen Kaur Ahuja)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

11/07/2012

Dr. Rajesh Kumar
Associate Professor,
SMCA, Thapar University, Patiala.
(Supervisor)

Countersigned By:

Dr. S.S. Bhatia
Professor & Head
SMCA, Thapar University, Patiala.

Dr. S.K. Mohapatra
Dean of Academic Affairs
Thapar University, Patiala.

Acknowledgement

It would not have been possible to write this thesis without the help and support of the kind people around me, to only some of whom it is possible to give particular mention here.

Thank God for the wisdom and perseverance that he has been bestowed upon me during this thesis, and indeed, throughout my life:"I can do everything through him who give me strength."

I am deeply grateful to my supervisor Dr. Rajesh Kumar, Associate Professor, School of Mathematics and Computer Applications (SMCA) for his able guidance, first hand experience, motivation, support and sympathetic attitude to write and complete my thesis. He has qualities to be an ideal guide. The way he helped me in my thesis work, requires no elaboration. Hence he will always prevail upon my remembrance.

I wish to express my sincere thanks to Dr. S.S Bhatia, Professor & Head of SMCA, Thapar University, Patiala for providing me University's resources and facilities necessary to carry out this research work.

I may not forget to pay my special thanks to Mr. Neeraj Juneja, Assistant Professor, Galgotias University, Noida.

I would also like to thank my family for their unconditional support. I thank my parents, who were always their supporting me and encouraging me with their best wishes. Without their support, this work would have been very difficult for me to tackle.

Harleen Kaur

Harleen Kaur Ahuja

Contents

Abstract

For a large software project, testing has a deep influence on the overall acceptability and quality of the final software as many software fails due to pure quality. Software metrics play a very important role in improving the software quality because by analyzing the metric data, we can forecast the quality of the object oriented system. Testability has been identified as the key factor of the software quality. Testability is an external software attribute that assesses the complexity and effort required for testing software. In this thesis, we estimate the Testability of one of the version of industrial project “The Health Watcher” system by using fuzzy logic based approach in which we used CK metrics as an input, and testability as output is calculated.

Chapter

1

Introduction

These days, better quality of the software has become an essential feature. For a large software project, testing has a deep influence on the overall acceptability and quality of the final software. Having quantitative data on testability is of immediate use in the software development process. Software testability can be easily predicted by using the metrics data. The tester can use this data to find out which piece of code is to be focused more during testing. The software engineer can use the testing data to refractor his code and build the class that can be reused and easily maintained in future.

1.1 Object Oriented System

Object orientation has rapidly become accepted as the preferred paradigm in industrial software development environments for large-scale system design. This technology offers support to provide software product with higher quality and lower maintenance costs [1]. Classes in Object Oriented System(OOS) provides an excellent structuring mechanism that allows a system to be divided into well designed units, which may then be implemented separately. An object-oriented system is composed of objects. The behavior of the system results from the collaboration of those objects. Collaboration between objects involves them sending messages to each other. Sending a message differs from calling a function in that when a target object receives a message, it decides on its own what function to carry out to service that message. The same message may be implemented by many different functions, the one selected depending on the state of the target object.

1.2 Software Metrics

Software engineering is a profession dedicated to analysis, designing, implementing and modifying software so that we develop software of high quality, and fast to build. Estimating software quality is an important task in the software development as many of the software fail due to poor quality. Software quality is the attribute to measure the software characteristics. Software metrics mainly design metrics plays an important role in helping developers to improve software quality attributes and supporting various software engineering activities [2-5].

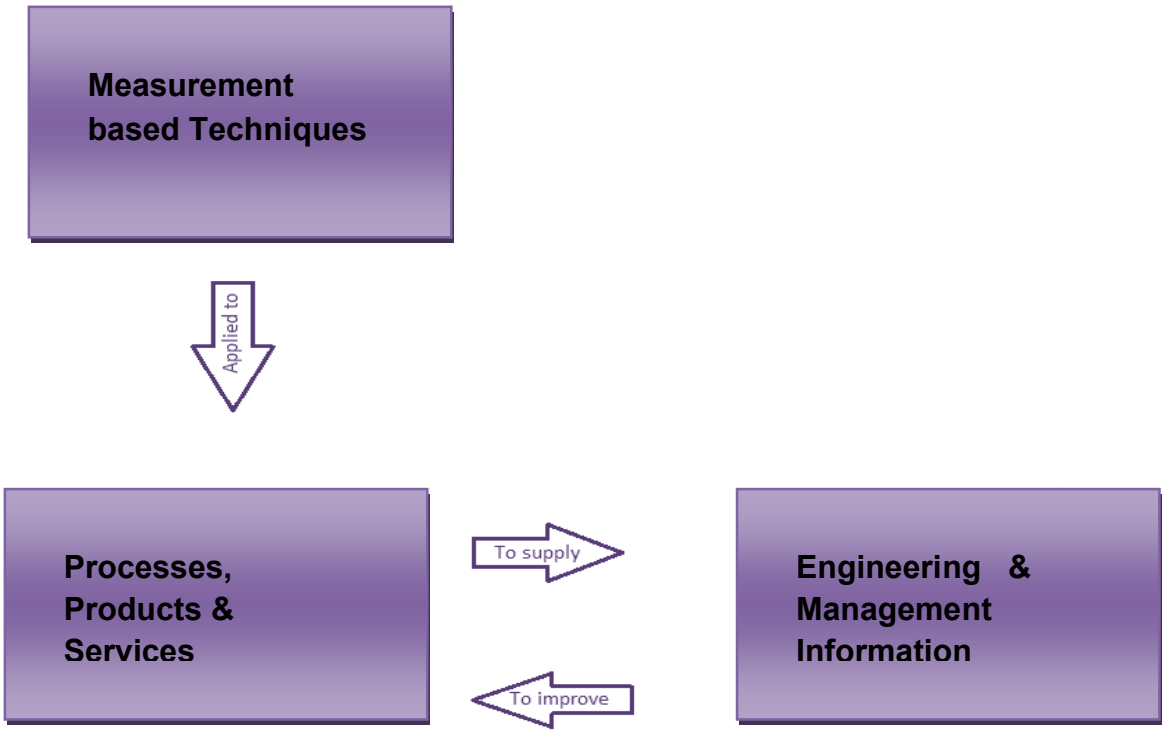

Fig. 1.1 Block Diagram of Software Metrics

Software metric is defined as “The continuous application of measurement-based techniques to the software development process and its products to supply meaningful and timely management information, together with the use of those techniques to improve that process and its products”.

In particular, metrics can be used to predict software testability and better manage the testing effort. Metric are the quantitative measure of the degree to which a system, component, or process possesses a given attribute. Software metrics can be used to measure different characteristics of a software system or software development process. Software Metrics are all about measurements which, in turn, involve numbers to make things better, to improve the process of developing software and to improve all aspects of the management of that process. Software metrics are essential to plan, predict, monitor, control, evaluate, products and processes. The main goal of the software metrics is to reduce costs, improve quality, reduce testing effort, many reusable fragments, to better understand the quality of the product and the program [6].

In this thesis, we used the metric suite proposed by Chidamber and Kemerer also called CK metric suite. CK have proposed six metrics, that are Weighted Methods per Class(WMC),Response for a Class(RFC),Coupling Between Objects(CBO),Depth Inheritence Tree(DIT),Number of Children(NOC) and Lack of Cohesion of Methods(LCOM). These metrics have been used in many NASA projects and empirically supported guidelines have been developed for their interpretation [7]. These metrics are used to measure the various software quality attributes like testability, reliability, usability, maintainability, understandability, complexity etc. Out of the six metrics proposed by CK, five metrics are chosen that reflects the testability of the software that are WMC, RFC, DIT, CBO and LCOM.

1.3 Testability

Testability is a distinct and the most important software quality characteristics. It expresses the affect of software structural and semantic on the effectiveness of testing following certain criterion. The effectiveness of testing decides the quality of released

software. According to IEEE Standards [8], testability is defined as the degree to which a system or component facilitates the establishment of test criteria and performance of tests to determine whether those criteria have been met. ISO [9] defines it as attributes of software that bear on the effort needed to validate the software product. Voas *et al*. [10] defines software testability as the probability that a piece of software will fail on its next execution during testing, provided it contains a fault. This fault sensitivity is obtained by multiplying the probabilities that

(i) the location containing the fault is executed;

(ii) the fault corrupts the program's state; and

(iii) the corrupted state gets propagated to the output.

High fault sensitivity indicates high testability and vice versa. Software testability is a key aspect to allow the detection of difficulty to uncover defects in software. Software testability supports the testing process and facilitates the creation of better quality software. The goal of increasing the testability of software is not just to detect defects but more importantly, to detect defects as soon as they are introduced. Thus, reducing the cost and time to fix the bug and producing higher quality software for each build of the release.

Software testability is affected by many different factors, including the required validity, the process and tools used and the representation of the requirements.

1.4 Fuzzy Logic

Fuzzy logic can coordinate two form of knowledge i.e. objective knowledge (that exists in the mathematical form) and subjective knowledge (that exists in linguistic form, usually impossible to quantify) in a logical way. Fuzzy logic is the methodology to solve problems, which are too complex to be understood quantitatively. Fuzzy systems try to behave just like the processes of the brain with a rule base. The basic concept is inspired by the human processes where the decisional criteria are not clear cut, but blurred and it is difficult to find objective to make the decisions more precise and clear. Similarly like the human brain, fuzzy logic accept imprecise data and vague statements and provides decisions. Fuzzy logic is the generalization of simple two valued logic

which treats “truth” as a continuous quantity ranging from 0 to 1. A simple two valued logic having two truth values true and false is sometimes inadequate when describing human reasoning. Fuzzy logic models the uncertainties of human language through the concept of partial true and partial false values between "completely true(1)" and "completely false (0)" through fuzzy sets in the interval [0,1].

In this thesis, we proposed a Fuzzy logic based approach for estimating testability for an object oriented system.

Chapter

2

Object Oriented Software Engineering

Object-Oriented Software Engineering (OOSE) is a software design technique that is used for software design in object-oriented programming. OOSE was developed by Ivar Jacobson in 1992. Software engineering has traditionally been an expensive and time-intensive process. Object orientation has rapidly become accepted as the preferred paradigm in industrial software development environments for large-scale system design. This technology offers support to provide software product with higher quality and lower maintenance costs , enabling software engineers to produce more flexible and easily maintainable applications. OOS is composed of objects. The behavior of the system results from the collaboration of those objects. Classes in object oriented system provide an excellent structuring mechanism. In other words, classes and objects are the basic building blocks used by an object-oriented system to analyze or design a complex software system. These concepts have been elaborated in this chapter.

2.1 Objects

Objects are the building blocks in an object oriented system. All entities except part of a message, comments and certain punctuation symbols are objects. They may represent a person, a place, a bank account, a table of data or any item that the program has to handle. An object is made up by the combination of data and methods(functions) i.e data and code to manipulate that data.

Object = Data + Methods

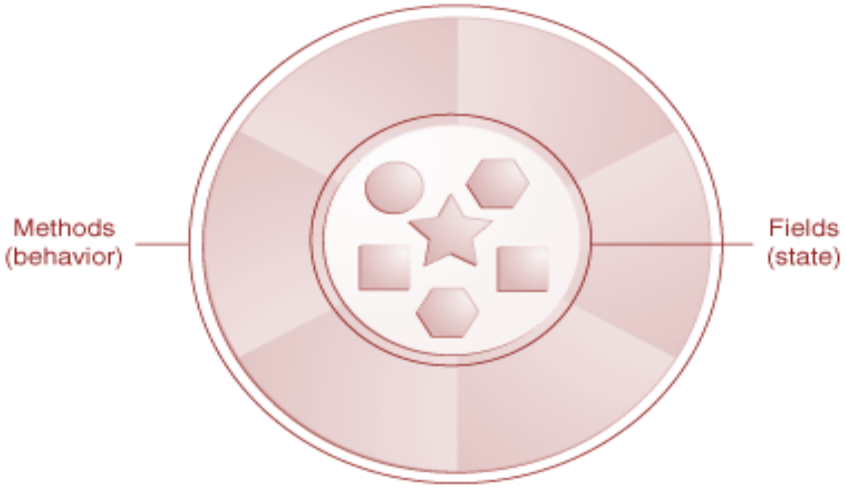

Fig. 2.1 A Software Object

Object is an entity that can store data and send and receive messages. An object can communicate with other objects using messages. An object passes a message to another object, which results in the invocation of a method. Objects then perform the actions that are required to get a response from the system. Objects are characterized by attributes and by the way they behave when the messages are sent to them. All real world objects share two characteristics i.e. state and behaviour. We can take an example of a Dog. Dogs have state (name, color, breed, hungry) and behavior (barking, fetching, wagging tail). Identifying the state and behavior for real-world objects is a great way to begin thinking in terms of object-oriented programming. Software objects are conceptually similar to real world objects. An object stores its state in fields, and it exposes its behaviours through its methods. An object consists of a limited amount of memory which contains other objects(data and methods). They are encapsulated together as a unit and are accessible to that object only. All objects belong to the same class. Example: Objects can be numbers (such as 4, 9, 42, 5.12) , or strings (like: ‘ this is an object’).

2.2 Classes

A class is a collection of objects that have common properties, operations and behaviors. A class is a combination of state (data) and behavior (methods). In object-oriented languages, a class is a data type, and objects are instances of that data

type. In other words, classes are prototypes from which objects are created or we can say that a class models the state and behavior of the real world objects. In the real world, we often find many individual objects all of the same kind. There may be thousands of other bicycles in existence, all of the same make and model. Each bicycle was built from the same set of blueprints and therefore contains the same components. In object-oriented terms, we say that a bicycle is an instance of the class of objects known as bicycles.

In another example, we may design a class Human, which is a collection of all humans in the world. Humans have state, such as height, weight, and hair color. They also have behavior, such as walking, talking, eating. All of the state and behavior of a human is encapsulated (contained) within the class human.
This can be represented as:

Table 2.1 :Example of a Class

```
public class Human{
int height;
String haircolor;
int age;
void walking(){
        }
void talking{
        }
void eating{
        }
```

2.3 Object Oriented Approach

In today's scenario, the process improvement is very much required in the software development field. The focus on process improvement has increased the demand for an improved approach *i.e*. Object Oriented approach and the software measures that are the metrics which manage the process. OO approach uses objects as its fundamental blocks. Moreover, this approach design classes and group similar classes into packages which make large software easier to manage. Object-oriented technology cuts development time and overhead, leading to faster time to market and significant competitive advantage, enabling software engineers to produce more flexible and easily maintainable applications.

.

2.4 Benefits of Object Oriented Software Engineering

Object orientation has rapidly become accepted as the preferred paradigm in industrial software development environments for large-scale system design. This technology offers support to provide software product with higher quality and lower maintenance costs [1]. The benefits of OOSE are:

Reduced Maintenance

The primary goal of object-oriented development is the assurance that the system will enjoy a longer life while having far smaller maintenance costs. Because most of the processes within the system are encapsulated, the behaviors may be reused and incorporated into new behaviors. Hence, objects can be maintained separately, thereby, locating and fixing problems easier.

Real-World Modeling

Object-oriented system tend to model the real world in a more complete fashion than do traditional methods. Objects are organized into classes of objects, and associated with behaviors. The model is based on objects, rather than on data and processing. Since

software objects model real world objects, so the complexity is reduced and the program structure is very clear.

Improved Reliability and Flexibility

Object-oriented system promise to be far more reliable than traditional systems, primarily because new behaviors can be "built" from existing objects. Because objects can be dynamically called and accessed, new objects may be created at any time. The new objects may inherit data attributes from one, or many other objects. Behaviors may be inherited from super-classes, and novel behaviors may be added without effecting existing systems functions.

High Code Reusability

When a new object is created, it will automatically inherit the data attributes and characteristics of the class from which it was created. The new object will also inherit the data and behaviors from all super classes in which it participates. When a user creates a new type of a widget, the new object behaves "wigitty", while having new behaviors, which are defined to the system.

Easy Modifiability

It is easy to make minor changes in the data representation or the procedures in an OO program. Changes inside a class do not affect any other part of a program, since the only public interface that the external world has with the class is through the use of methods.

Chapter

3

Software Quality

Software Engineering is a discipline whose aim is the production of quality software, software that is delivered on time, within budget, and that satisfies its requirements. Software Engineering involves the study of the means of producing high quality software products with predictable costs and schedules. One of the major goal in software engineering is to control the software development process, thereby controlling costs and schedules, as well as the quality of the software products. Software metrics can provide a quantitative means to control the software development process and the quality of software products. In the context of software engineering, software quality measures how well software is designed (quality of design), and how well the software conforms to that design (quality of conformance). It is often described as the 'fitness for purpose' of a piece of software. ISO 9126 defines Software quality as " The totality of functionality and features of a software product that bear on its ability to satisfy stated or implied needs". Also, Software quality relates to the degree of which a system, system component or process meets a specified requirements or alternatively, a customer or user's needs or expectations.

3.1 Classification of Software Quality

The ISO/IEC standard describes a Software Quality model[9], which is a standard for evaluating Software quality.

Software quality model categorizes software quality into six characteristics which are further sub-divided into sub-characteristics. Fig. 3.1 shows a block diagram of ISO quality model.

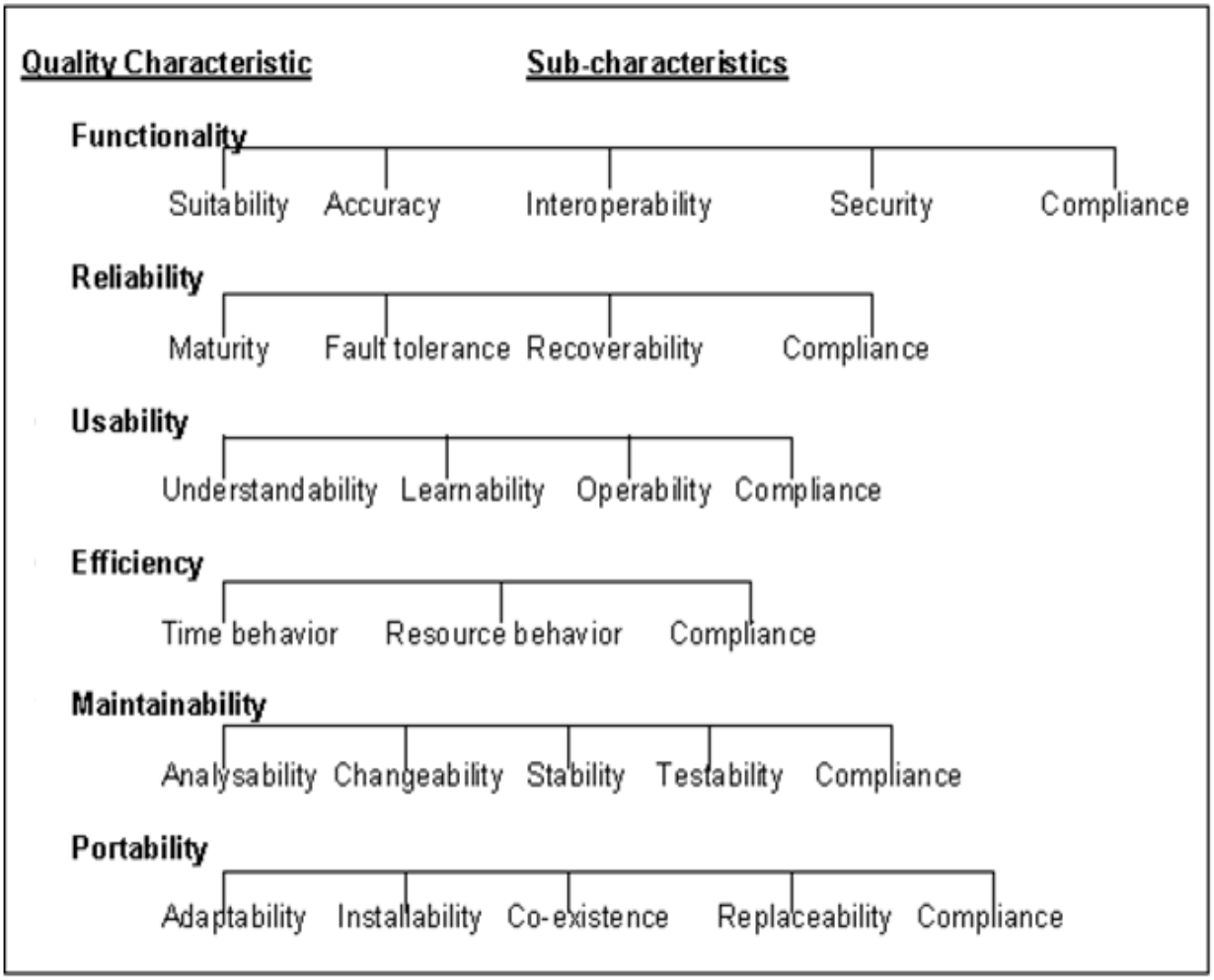

Fig. 3.1. ISO 9126 Quality Model

The six main quality characteristics are:

a. Functionality
b. Reliability
c. Usability
d. Efficiency
e. Maintainability
f. Portability

a. Functionality

Functionality is the set of attributes that bear on the existence of a set of functions and their specified properties. The functions are those that satisfy stated or implied needs. Functionality is the essential purpose of any product or service. For certain items, this is relatively easy to define. For example, a ship's anchor has the function of holding a ship at a given location. The more functions a product has, the more complicated it becomes to define it's functionality *e.g.* an ATM machine. The sub-characteristics of Functionality are as follows:

Suitability : This is the essential functionality characteristic and refers to the appropriateness of the functions of the software.

Accurateness : This refers to the correctness of functions.

Interoperability : This sub-characteristic concerns the ability of a software component to interact with other components or system.

Compliance : This sub-characteristic addresses the complaint capability of software.

Security : This relates to unauthorized access to the software functions.

b. Reliability

Reliability is defined as the capability of the software to maintain its level of performance under stated conditions for a stated period of time. It is also defined as the probability of failure-free operation. For example, if the network goes down for 20 seconds, then comes back, the system should be able to recover and continue functioning. The sub-characteristics of Reliability are :

Maturity : This sub-characteristic concerns the frequency of failure of the software.

Fault Tolerance : This refers to the ability of the software to withstand and recover from component, or environmental failure.

Recoverability : The ability to bring back a failed system to full operation, including data and network connections is called Recoverability.

c. Usability

Usability is an attempt to define user friendliness. It is the ease of use for a given function. It can be measured in terms of :

- Physical and intellectual skill required to learn the system
- Time required to become moderately efficient in the use of the system
- The net increase in productivity over the system it replaces
- A subjective assessment of users attitudes towards the system

For example, a function of an ATM machine is to dispense cash as requested. Placing common amounts on the screen for selection, i.e. Rs200, Rs500, Rs800 etc, does not impact the function of the ATM but addresses the Usability of the function. The sub characteristics of Usability are:

Understandability : This sub-characteristic determines the ease of which the system functions can be understood.

Learnability : This is the ability to learn how to use the system.

Operability : This is the ability of the software to be easily operated by a given user in a given environment.

d. Efficiency

Efficiency is the set of attributes that bear on the relationship between the level of performance of the software and the amount of resources used, under stated conditions. The amount of disk space, memory, network etc. provides a good indication of this characteristic. For example, the usability of a system is influenced by the system's performance, it means, if a system takes 3 hours to respond, the system would not be easy to use although the essential issue is a performance or, efficiency characteristic. The sub-characteristics of Efficiency are:

Time Behavior : This characterizes the response time for a given throughput i.e. transaction rate.

Resource behavior : This characterizes the resources used i.e. memory. cpu, disk and network usage.

d. Maintainability

Maintainability addresses the ability to identify and fix a fault within a software component. Maintainability is impacted by code readability or complexity as well as modularization. Anything that helps with identifying the cause of a fault and then fixing the fault is the concern of maintainability. Software has a high degree of maintainability, if it is easy to understand, enhance, and correct. Sub-characteristics of Maintainability are as follows:

Analyzability : This is the ability to identify the root cause of a failure within the software.

Changeability : This means the amount of effort to change a system.

Stability: This characterizes the sensitivity to change of a given system that is the negative impact that may be caused by system changes.

Testability: This sub-characteristic characterizes the effort needed to verify (test) a system change.

f. Portability

Portability is the set of attributes that bear on the ability of software to be transferred from one environment to another. This characteristic refers to how well the software can adapt to changes in the environment or with its requirements. The sub-characteristics of Portability are as follows:

Adaptability: This characterizes the ability of the system to change to new specifications or operating environments.

Installability: This sub-characteristic characterizes the effort required to install the software.

Conformance: This is similar to complaince of functionality, but this characteristic relates to portability. *e.g*., Open SQL conformance which relates to portability of database used.

Replacability: This characterizes the plug and play aspect of software components, that is, how easy is it to exchange a given software component within a specified environment.

Among all the characteristics and sub-characteristics of software quality, testability has always being the broad concept to measure and evaluate the software accurately. Testability is a software quality characteristic which is of major relevance for test costs and software dependability. Improving software testability is clearly an important objective in order to reduce the number of defects that result from poorly designed software.

3.2 Testability Metrics

IEEE Standards define testability as the degree to which a system or component facilitates the establishment of test criteria and performance of tests to determine whether those criteria have been met. Software testability is a key aspect to allow the detection of difficulty to uncover defects in software. Software testability supports the testing process and facilitates the creation of better quality software.

To measure testability, a number of software metrics are used . Chidamber and Kemerer proposed six metrics[11] for an object-oriented design, out of which five affect testability of the software system. These are :

a. **Lack of Cohesion of Methods(LCOM)**
b. **Coupling Between Objects(CBO)**
c. **Depth Inheritance Tree(DIT)**
d. **Weighted Methods per Class(WMC)**
e. **Response For a Class(RFC)**

a. Lack of cohesion of methods (LCOM)

The LCOM metric measures the lack of cohesion of a class. The cohesion of a class is characterized by, how closely the local methods are related to the local instance variables in the class. LCOM is defined as the number of different methods within a class with reference to a given instance variable. It measures the degree of similarity of methods by instance variable or attributes. Consider a class C with n methods M_1 , M_2 ,……M_n. Let (I_j)= set of all instance variables used by method Mi.

Let P={(I_i,I_j)| $I_i \cap I_j = \emptyset$} and Q = {(Ii.Ij)| $I_i \cap I_j \neq \emptyset$} then

LCOM = |P|- |Q|, if |P|> |Q|

= 0 ,otherwise

A positive high value of LCOM(*i.e.* lack of cohesion or low cohesion) implies that classes are less cohesive and increases complexity thereby, increasing the likelihood of errors during the development process, whereas high cohesion implies simplicity. So, a low value of LCOM is desirable for better testability[11].

b. Coupling between Objects(CBO)

Coupling between object classes (CBO) is a count of the number of other classes to which a class is coupled. A class is coupled to another class, if it uses its methods or instance variables. The more independent a class is, the easier it is to reuse in another application. The larger the number of couples, the higher the sensitivity for changes in other parts of the design and therefore maintenance, and hence testing is more difficult. Strong coupling complicates a system since a class is harder to understand, change or correct by itself, if it is interrelated with other classes[12].

c. Depth Inheritance Tree (DIT)

Depth Inheritance Tree is defined as the maximum length from the node to the root of the tree and is measured by the number of ancestral classes. The deeper a class is in the hierarchy, the greater the number of methods it is likely to inherit, making it more complex to predict its behavior.

d. Weighted Methods per Class(WMC)

WMC is the sum of the complexities of all the methods in a class. According to this metric, if a Class C, has n methods and C_1, C_2 …C_n be the complexity of the methods, then, $WMC(C)= C_1 + C_2 +… +C_n$.

or,

$$\mathrm{WMC} = \sum_{i=1}^{n} C_i$$

Mc Cabe's cyclomatic complexity metric is chosen for calculating the complexity values of the methods of a class. It is a count of number of test cases that are needed to test the method comprehensively. As a software metric , cyclomatic complexity is expressed as

V(G)= P + 1 , where P is the number of predicate nodes(tests/ conditions)

A method with a low cyclomatic complexity is generally better because it leads to decreased testing and increased understandability.

e. Response for a class(RFC)

RFC is the count of the set of all methods that can be invoked in response to a message to an object of the class or by some method in the class.

RFC = |RS| where RS is the response set for the class. The response set for the class can be expressed as

RS = {M} + { R□}

where {M}= set of all methods in a class and

{R□} = set of all methods called by method i

If a large number of methods can be invoked in response to a message, the testing and debugging of the class becomes difficult in view of the fact that it requires a better level from the test engineer side [13].

3.3 Case study

In our case study, we have considered an industrial project "**The Health Watcher**" system and calculated the values of **Testability Metrics** in one of its version. It consist of 91 components and the code of all these components is written in **Java**.

The Health Watcher (HW) system[14] is a web based application that allow citizens to register complaints regarding issues in health care institutions. The system is also used to notify people about important information regarding the Health System. A citizen can access the system, and make their complaint or ask information about the health services. In the event of a complaint being made, it will be registered on the system and

addressed by a specific department. This department will be able to handle the complaint in an appropriate manner and return a response when complaint has been dealt with. This response will be registered on the system and available to be queried.
In the HW system, complaints are registered, updated and queried through a Web client implemented using Java Servlets, represented by the components in the view layer. Accesses to the HW services are made through the *IFacade* interface, which is implemented by the *HealthWatcherFacade*. This facade works as a portal to access the business collections, such as *ComplaintRecord* and *EmployeeRecord*. Records access the data layer using interfaces, like *IComplaintRep*, which decouple the business logic from the specific type of data management in use. For example, Fig.3.2 shows a *ComplaintRep* class that implements a repository for a database. Almost all components must deal with the exception handling concerns. Fig.3.2 shows an Object Oriented HW Architecture System .

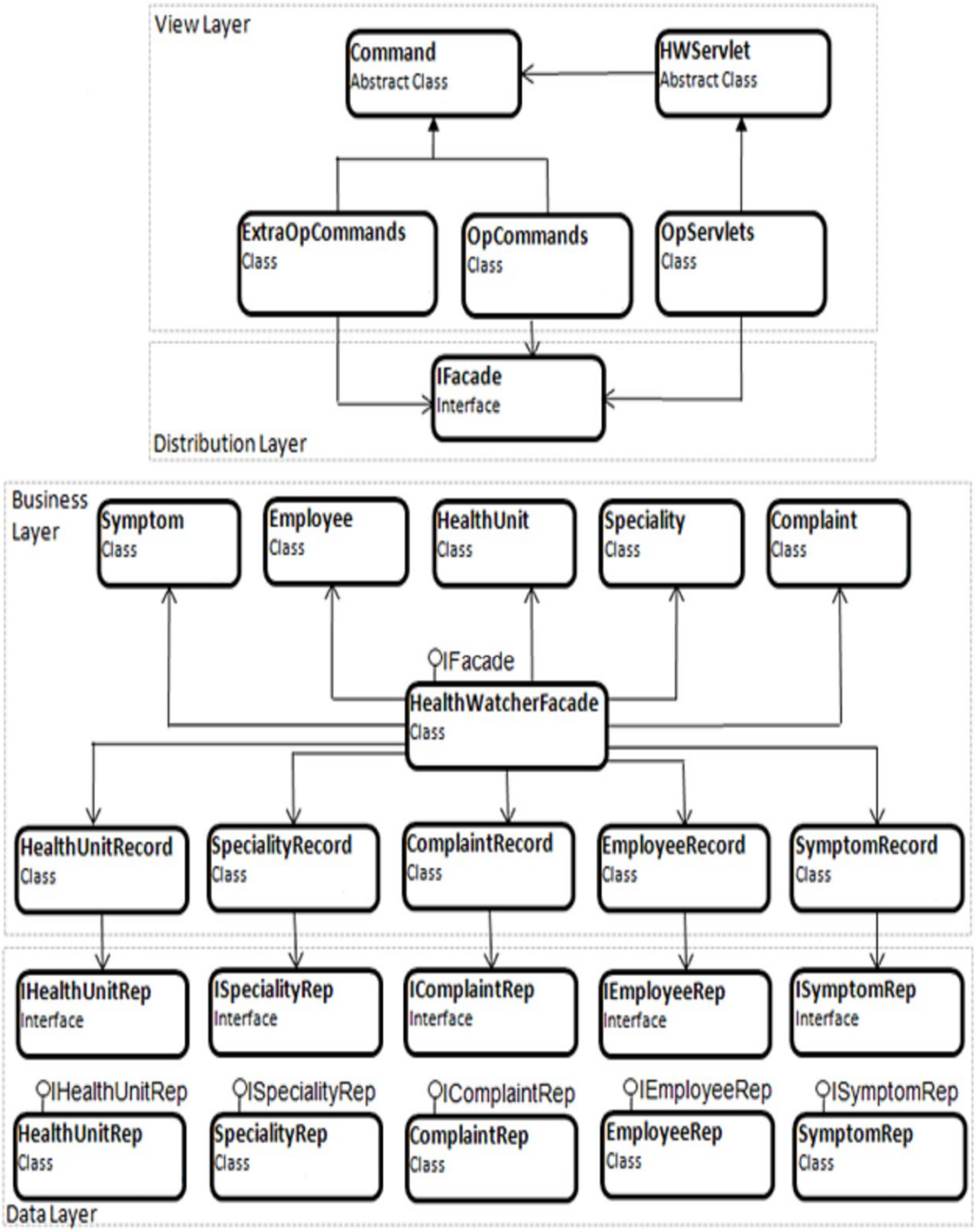

Fig. 3.2 Object Oriented Health Watcher (HW) Architecture

These CK testability metrics discussed before are be measured manually and using the automated tool **JHawk 5.0**.

Since its release in 1999, JHawk has been a leader in the provision of Software Metrics to Java developers. JHawk has proved itself in many different areas and its customers includes from Fortune 500 companies to Academic institutions, from Banking to Telecommunications companies and across the globe from Norway to Brazil, and from the US to China. JHAWK 5.0 tool is used to measure the inherent quality of software and java program. The individual class file and the java program can be measured. The process of taking code and analysing it is called 'static analysis'. Sensible decisions can be made based on the results of analysis of the code. The metrics are measured at system's package, class level or method level on java code.

In this thesis, we have used Jhawk tool with the eclipse plug-in to collect the metric data. For this, we have to first copy the contents of Jhawk5Plugin into the Eclipse Plug-in. After restarting, Eclipse JHawk is ready to use. To analyse the code with JHawk, simply go to the Project explorer window and select the source code that you wish to analyze (Projects, Packages, Classes). You can multiply select items if you wish. Right click on the selected items and look for the Analyze Project, Package or File. To view metrics of source program, we have to enable JHawk Metrics view from the menu list. It will display the object oriented metrics for the code. A metric view is shown in the Fig. 3.3 on the next page.

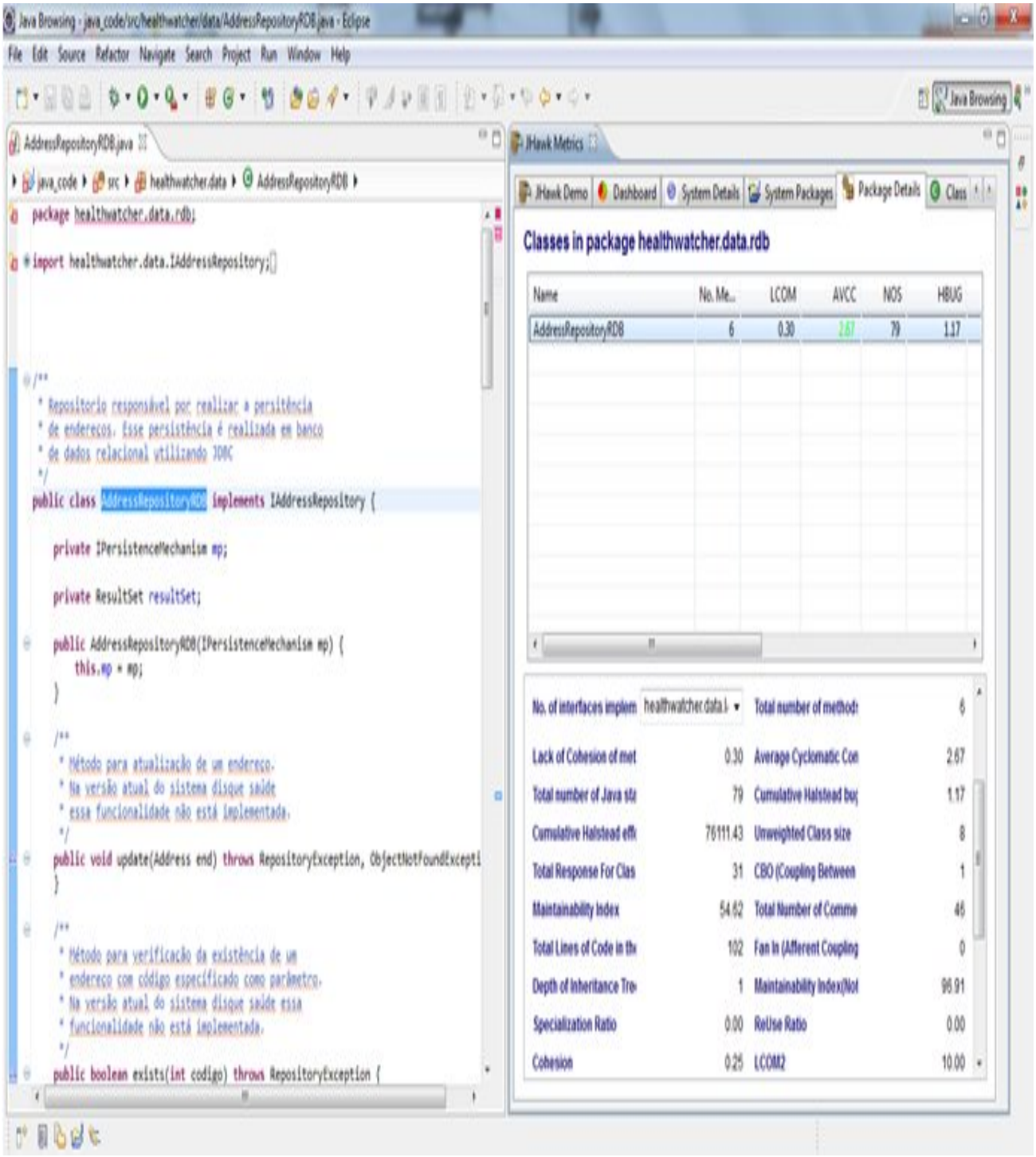

Fig. 3.3 Metric View of AddressRepositoryRDB.java

The values of all the five input metrics are computed for all the 91 components with the help of JHawk tool and two metrics i.e. RFC and WMC are also measured manually.

These input CK metric values and testability values(output) which are evaluated through fuzzy logic system are shown in table 4.1 in the next chapter .
Software testability can be easily predicted by using the metrics data. The tester can use this data to find out which piece of code to be focused during testing. The software engineer can use the testing data to refractor his code and build the class that can be reused and easily maintained in future.

Bruntinket *et al*. [15] identified several qualitative and quantitative factors affecting testability of software. Qualitative factors include testing criteria, implementation, documentation, test suites and test tools. They related the testability of a system to the number of test cases required to test it and to the effort required to develop each individual test case. They found a correlation between class level metrics such as number of methods and test level metrics such as the number of test cases and the number of lines of code per test class. However, no correlation was found between Inheritances related metrics (*e.g.* depth of inheritance tree) and the proposed Testability Metrics.

Binder [16] listed a number of simple metrics to assess testability, *i.e.* lack of cohesion in methods, percentage of non-overloaded calls, percentage of dynamic calls, and depth of inheritance tree.However, it did not provide any empirical evidence that there is a correlation between the suggested metrics and testability. He took Inheritance related metrics to propose Testability Metrics but failed to include Coupling concept.

Briand *et al*. [17] illustrated an approach where instrumented contracts are used to increase observability, diagnosability and testability of a program. But Briand did not explore the code structure to propose a testability measure.

Gupta *et al*. [12] use fuzzy techniques to combine four OO metrics values into a single overall value called testability index. The proposed approach has been evaluated on simple examples of Java classes.

Chapter

4

Estimation of Testability of an Object Oriented System

For large software systems such as Object oriented System, the overall acceptability and quality are judged by measuring the Testability of the software. While designing a software, Testability plays an important role as it assesses the complexity and effort require for testing software. A good measure for testability better manages the testing effort and time. Software testability can be easily predicted by using the metrics data. The tester can use this data to find out which piece of code to be focused during testing and he can refactor the code if required.

4.1 Methodology for Estimation of Testability of Object Oriented System

In this chapter, we proposed a Fuzzy Logic based approach for estimating Testability of Object Oriented System. To estimate Testability of an OO system, five OO metrics are chosen which affect Testability. A relationship between CK metrics and Testability has been established and viewed how these metrics affect testability. The chosen metrics are as follows:

- LCOM
- CBO
- DIT
- WMC
- RFC

It has been observed that any one of the above mentioned metrics cannot alone estimate the testability of an object oriented system, so all the five metrics are combined to estimate the testability of an OO system through the Mamdani Model of Fuzzy Logic approch.

Lo and Shi [18] have tried to find out a unified measure for testability. But their approach is erroneous due to several accounts. First, they have multiplied the testability derived from above factors. As these factors are parallel dependent or independent they cannot be multiplied to arrive on a single number for system's testability. Shortcoming at one place cannot diminish the testability of the whole system drastically. Moreover all the factors have different unit of measurement, therefore, multiplication cannot be used without normalization or without suggesting some weightage to each other. To handle all these shortcomings, Fuzzy Logic is an ideal choice. Secondly, they have used coupling to measure class testability, while it is a measure of interaction between classes not of class internals. Lastly, they have not suggested any measure of testability for the whole system. So, in our thesis we use fuzzy model to estimate the testability for an Object Oriented system as fuzzy model is the best choice to handle the doubtful, contradicting and divergent options.

4.2 Fuzzy Model

Fuzzy logic is the generalization of simple two valued logic which treats " truth" as a continous quantity ranging from 0 to 1. Fuzzy logic [19] models the uncertainties of human language through the concept of partial true and partial false values between "completely true(1)" and "completely false (0)" through fuzzy sets in the interval [0,1]. A fuzzy set is a set that allows its members to have different degree of membership, called **membership function** in the interval [0,1]. Fuzzy systems try to behave just like the processes of the brain with a rule base. Similarly like the human brain, fuzzy logic accept imprecise data and vague statements and provides decisions.

The fuzzy model consists of four modules. The first module is the fuzzification that transforms the crisp values into the fuzzy values. In the next step, the fuzzy values are inferenced as per the rules incorporated in knowledge base. These rules are supplied

by the domain expert(s). All the outputs obtained from the inference engine are integrated and finally defuzzied by the defuzzification module that transform the fuzzy output to crisp values.

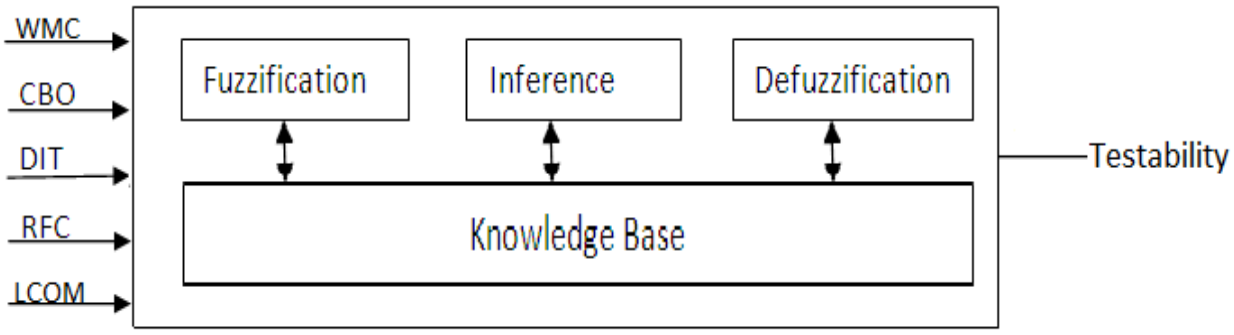

Fig. 4.1 A Fuzzy Model

4.3 Implementation through Fuzzy Logic

First all the crisp values of all the five inputs WMC,CBO,DIT,RFC and LCOM are taken for the output Testability in the Mamdani fuzzy model. All the inputs are partitioned into three Membership functions namely LOW, MEDIUM and HIGH. The output of the system Testability is specified by five Membership functions *viz*. VERY_LOW, LOW, MED, HIGH, VERY_HIGH . After output specification, the next step is rule base generation. In order to measure the software testability using the five input metrics having three MembershipFunctions each, 3^5 (two hundred forty three) rules have been defined. Now the degree to which the inputs belong to each of the appropriate fuzzy sets is determined.

Depending on a particular set of inputs, a rule is fired. The outputs of all fired rules are aggregated and are then defuzzified using centroid technique to get a crisp value of testability on scale of 0 to 1.Using Rule Viewer, the value of testability can be observed for a particular set of inputs using Matlab Fuzzy tool box[20].

Lower value of testability reflects bad testability, while higher value indicates good testability.

Following are some figures which shows the steps of a Fuzzy model:

Fuzzification of inputs and outputs of fuzzy logic in matlab software is shown in Fig. 4.2.

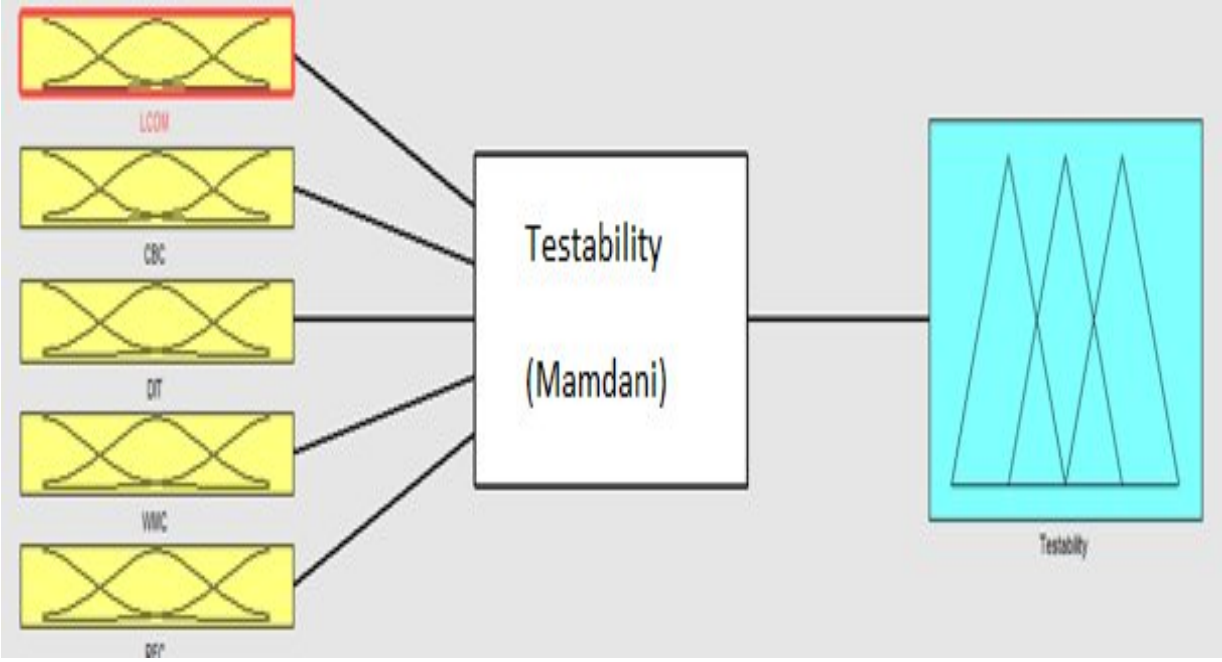

Fig. 4.2 Fuzzification

Among 11 membership fuctions available in Matlab, we considered Triangular Membership Function for our system. It is a three point function, defined by minimum , modal and a maximum value. In Table 4.1 and 4.2, the parameter values for all the five inputs and output are shown.

Table 4.1 : Parameter Values for Inputs

Input 1	'LCOM'
Range	[0 240]

NumMFs	3
MF1	'Low':'trimf',[0 55 100]
MF2	'medium':'trimf',[75 135 180]
MF3	'high':'trimf',[140 190 240]
Input 2	'CBC'
Range	[0 3]
NumMFs	3
MF1	'low':'trimf',[0 0.6 1.1]
MF2	'medium':'trimf',[0.75 1.35 1.95]
MF3	'high':'trimf',[1.65 2.45 3]
Input 3	'DIT'
Range	[0 2]
NumMFs	3
MF1	'low':'trimf',[0 0.4 0.85]
MF2	'medium':'trimf',[0.55 1 1.4]
MF3	'high':'trimf',[1.15 1.55 2]
Input 4	'WMC'
Range	[0 95]
NumMFs	3
MF1	'low':'trimf',[0 20 40]
MF2	'medium':'trimf',[25 47.5 68]
MF3	'high':'trimf',[52 74 95]
Input 5	'RFC'
Range	[0 113]
NumMFs	3
MF1	'low':'trimf',[0 22 45]
MF2	'medium':'trimf',[35 55 78]
MF3	'high':'trimf',[68 90 113]

Table 4.2 : Parameter Values for Output

Output	'Testability'
Range	[0 1]

NumMFs	5
MF1	'Very_Low':'trimf',[0 0.1 0.25]
MF2	'Low':'trimf',[0.12 0.25 0.4]
MF3	'Medium':'trimf',[0.25 0.45 0.7]
MF4	'High':'trimf',[0.55 0.68 0.82]
MF5	'Very_High':'trimf',[0.75 0.9 1]

In the next figure, Fig. 4.3, Membership Function of the input LCOM is shown.

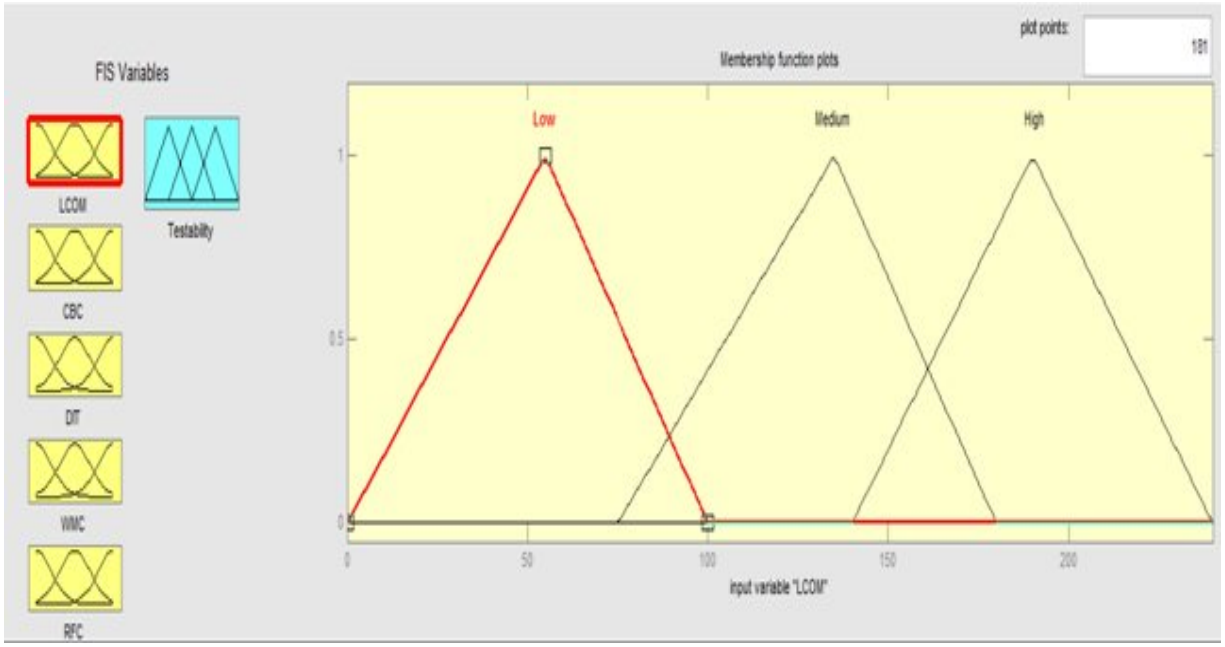

Fig. 4.3 Membership Function of Input Variable LCOM

The rules which were designed based on all the 243 combinations of the five inputs, helps to obtain the output i.e. testability using Rule Viewer, which is shown in Fig. 4.4 .

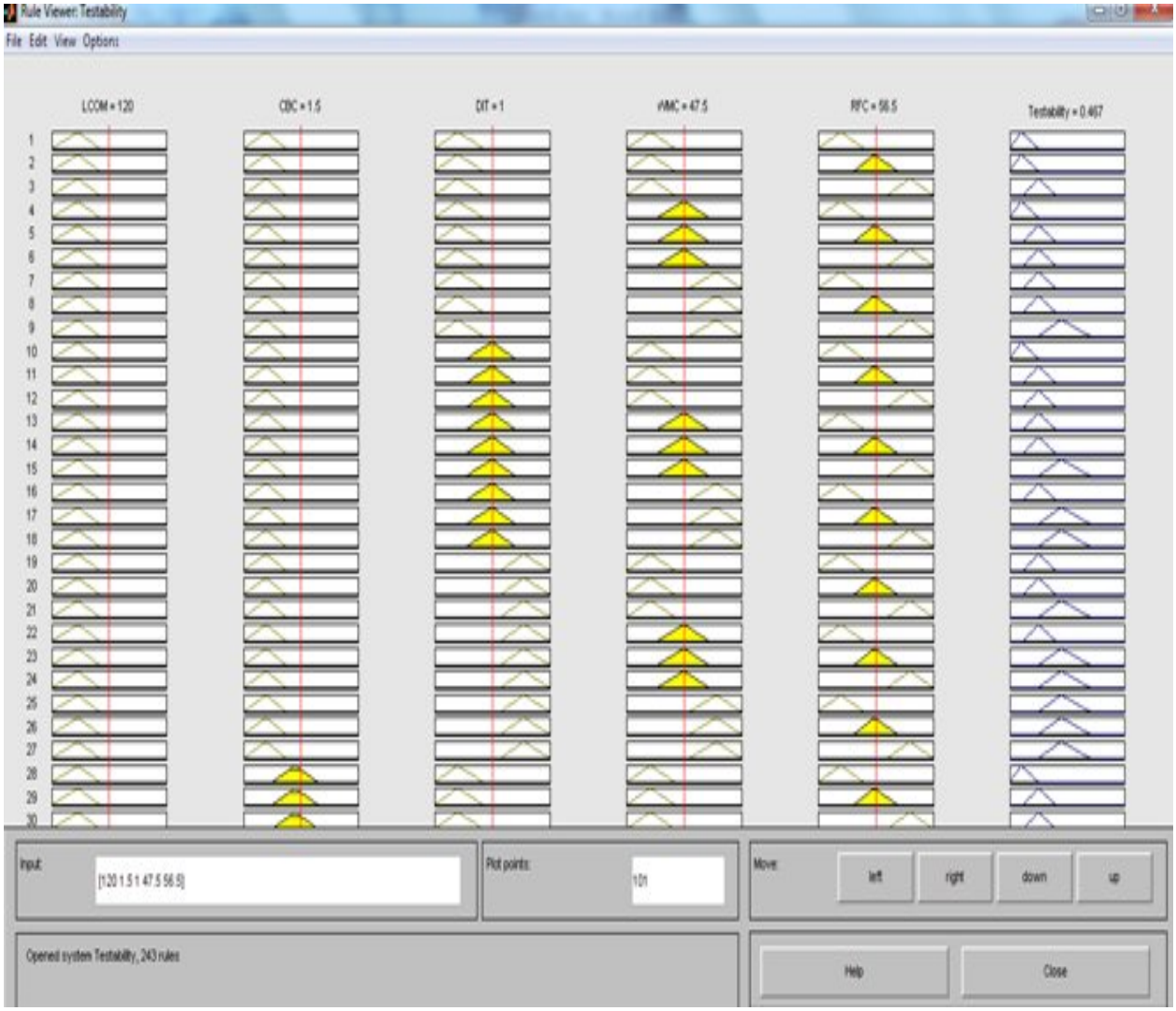

Fig. 4.4 Rule Viewer

The output Testability for different set of inputs can be observed either by changing the values in the 'Input' Box shown in the left bottom of Fig. 4.4 or by dragging the red line for each of the input.

The values of all the five input metrics (LCOM,CBO,DIT,WMC,RFC) as calculated by JHawk tool and the values of output (Testability) as estimated by Fuzzy model by giving the inputs to it are as shown in the Table: 4.3 .

Table 4.3 : Values of CK metrics (Input) and Testability (Output)

S.No.	Component	LCOM	CBO	DIT	WMC	RFC	Testability
1.	ComplaintRecord	0	1	1	10	21	0.5
2.	DiseaseRecord	1	0	1	3	5	0.5
3.	EmployeeRecord	0	1	1	5	10	0.5
4.	HealthUnitRecord	0	1	1	7	14	0.5
5.	MedicalSpecialityRecord	2	1	1	2	3	0.2
6.	HealthWatcherFacade	190	2	1	19	40	0.472
7.	ComplaintRepositoryArray	0	1	1	18	14	0.5
8.	DiseaseTypeRepositoryArray	0	1	1	18	16	0.5
9.	EmployeeRepositoryArray	0	0	1	17	14	0.5
10.	HealthUnitRepositoryArray	0	0	1	26	23	0.5
11.	SpecialityRepositoryArray	0	0	1	21	17	0.5
12.	SymptomRepositoryArray	0	0	1	19	14	0.5
13.	AddressRepositoryArray	10	1	1	16	31	0.198
14.	ComplaintRepositoryRDB	34	1	1	95	113	0.5
15.	DiseaseTypeRepositoryRDB	16	1	1	20	33	0.21
16.	EmployeeRepositoryRDB	10	1	1	15	20	0.198
17.	HealthUnitRepositoryRDB	2	0	1	42	39	0.5
18.	SpecialityRepositoryRDB	10	0	1	19	30	0.5
19.	IAddressRepository	5	0	1	5	5	0.5
20.	IComplaintRepository	6	0	1	6	6	0.5
21.	IDiseaseRepository	5	0	1	5	5	0.5
22.	IEmployeeRepository	5	0	1	5	5	0.5
23.	IHealthUnitRepository	8	0	1	8	8	0.5
24.	ISpecialityRepository	6	1	1	6	6	0.5
25.	ISymptomRepository	5	0	1	5	5	0.5
26.	Address	146	2	1	18	18	0.47
27.	AnimalComplaint	42	1	2	10	10	0.5
28.	Complaint	222	3	1	22	22	0.5

29.	DiseaseType	102	2	1	15	16	0.47
30.	FoodComplaint	114	0	2	16	16	0.5
31.	Situation	5	0	1	3	3	0.5
32.	SpecialComplaint	42	0	2	10	10	0.5
33.	Symptom	4	0	1	4	4	0.5
34.	Employee	26	1	1	8	9	0.218
35.	HealthUnit	16	2	1	11	11	0.259
36.	MedicalSpeciality	10	0	1	6	6	0.5
37.	Command	2	1	1	2	2	0.2
38.	CongigRMI	1	0	2	3	9	0.5
39.	GetDataForSearchByDiseaseType	1	0	2	4	15	0.5
40.	GetDataForSearchByHealthUnit	1	1	2	5	17	0.5
41.	GetDataForSearchBySpeciality	1	0	2	5	17	0.5
42.	InsertAnimalComplaint	1	0	2	5	18	0.5
43.	InsertEmployee	1	0	2	6	15	0.5
44.	InsertFoodComplaint	1	0	2	5	18	0.5
45.	InsertSpecialComplaint	1	0	2	5	17	0.5
46.	Login	0	2	2	6	15	0.5
47.	LoginMenu	0	1	2	4	12	0.5
48.	SearchComplaintData	1	0	2	20	46	0.5
49.	SearchDiseaseData	1	0	2	6	23	0.5
50.	SearchHealthUnitBySpeciality	1	0	2	5	16	0.5
51.	SearchSpecialityByHealthUnit	1	0	2	5	16	0.5
52.	UpdateComplaintData	1	1	2	4	22	0.5
53.	UpdateComplaintList	1	0	2	7	19	0.5
54.	UpdateComplaintSearch	1	2	2	10	20	0.5
55.	UpdateEmployeeData	1	0	2	5	18	0.5
56.	UpdateEmployeeSearch	1	0	2	5	15	0.5
57.	UpdateHealthUnitData	1	0	2	4	14	0.5
58.	UpdateHealthUnitList	1	0	2	6	15	0.5

59.	UpdateHealthUnitSearch	1	1	2	5	15	0.5
60.	HWServlet	4	1	2	3	4	0.5
61.	ServletWebServer	0	0	2	3	9	0.5
62.	IFacade	16	0	1	16	16	0.5
63.	Constants	1	0	1	1	1	0.5
64.	ConcurrencyManager	0	0	1	7	12	0.5
65.	IIteratorRMITargetAdapter	2	0	2	2	2	0.5
66.	IteratorRMISourceAdapter	0	1	1	19	15	0.5
67.	IteratorRMITargetAdapter	0	0	1	4	4	0.5
68.	CommunicationException	1	0	2	1	1	0.5
69.	ExceptionMessages	0	1	1	0	0	0.5
70.	InsertEntryException	5	0	2	5	5	0.5
71.	InvalidDateException	2	0	2	2	2	0.5
72.	InvalidSessionException	2	0	2	2	2	0.5
73.	ObjectAlreadyInsertedException	1	0	2	1	1	0.5
74.	ObjectNotFoundException	1	1	2	1	1	0.5
75.	ObjectNotValidException	1	0	2	1	1	0.5
76.	PersistenceMechanismException	1	0	2	1	1	0.5
77.	PersistenceSoftException	1	0	2	1	1	0.5
78.	RepositoryException	1	0	2	1	1	0.5
79.	SituationFacadeException	1	0	2	1	1	0.5
80.	TransactionException	1	0	2	1	1	0.5
81.	UpdateEntryException	4	0	2	4	4	0.5
82.	IPersistenceMechanism	7	0	1	7	7	0.5
83.	Persistencemechanism	36	1	1	41	37	0.334
84.	ConcreteIterator	7	0	1	7	7	0.5
85.	Date	99	1	1	76	48	0.47
86.	Functions	5	0	1	10	7	0.5
87.	HTMLCode	97	1	1	21	32	0.241
88.	IteratorDsk	4	2	1	4	4	0.26

89.	Liberary	7	2	1	14	19	0.26
90.	LocalIterator	4	0	1	4	4	0.5
91.	Schedule	26	1	1	30	22	0.219

As per the Output obtained for all the components, we are able to see that most of the components are having Testability value as **0.5**. This indicates that most of them are having reasonably good value of testability. Hence, Testability of HW system on the whole is reasonably good.

Chapter

5

Conclusion and Future Scope

Object Oriented Software Engineering (OOSE) is a branch of software engineering which emphasis on object oriented approach. Software engineering has traditionally been an expensive and time-intensive process. Object oriented analysis and design is the principal industry-proven methodology that answers the call for a more cost-effective, faster way to develop software and systems. Object-oriented technology cuts development time and overhead, leading to faster time to market and significant competitive advantage, enabling software engineers to produce more flexible and easily maintainable applications.

Software testability is a key aspect to allow the detection of difficult to uncover defects in software. Software testability supports the testing process and facilitates the creation of better quality software. The goal of increasing the testability of software is not just to detect defects but more importantly, to detect defects as soon as they are introduced. Thus, reducing the cost and time to fix the bug and producing higher quality software for each build of the release.

Among all the categories of software quality, testability has always being the broad concept to measure and evaluate the software accurately. Testability is a software quality characteristic that is of major relevance for test costs and software dependability. Improving software testability is clearly an important objective in order to reduce the number of defects that result from poorly designed software.

In this thesis, we have calculated the testability of one of the version of the industrial project “The Health Watcher” software by taking five CK metrics as input and Mamdani

Fuzzy Model of Matlab is used to get the Teatability (output). The values of all the input metrics are calculated using JHawk open source tool.

In future, testability of some more of the Industrial Project, such as, Mobile Media *etc*., can be calculated on the similar lines.

Further, some more quality metrics can also be explored.

References

[1] R. A. Khan and K. Mustafa, Metric Based Testability Model for Object Oriented Design, ACM SIGSOFT Software Engineering Notes, Volume 34, Issue 2, pp : 1-6, March 2009 .

[2] V. R. Basili, L. C. Briand and W. Melo, A Validation of Object oriented Design Metrics as Quality Indicator, IEEE Transactions on Software Engineering, Volume 22, Issue 10, pp : 751- 761, October 1996 .

[3] N. Fenton, S.L. Pfleeger, Software Metrics: A Rigorous and Practical Approach, PWS Publishing Company, 1997.

[4] R.S. Pressman, Software Engineering, A Practitioner's Approach, McGraw Hill, 6th Edition, New York, 2005 .

[5] I. Sommerville, Software Engineering, Addison Wesley, 9th Edition, 2011.

[6] P. E. Linda, V. M. Bhashni and S. Gomathi, Metrics for Component based measurement Tools, International Journal of Scientific and Engineering Research, Volume 2, Issue 5, ISSN 2229-5518, May 2011.

[7] L. H. Rosenberg, R. Stapko and A. Gallo, Risk Based Object Oriented Testing, NASA GSFC SATC NASA, Unisys SATC NASA, Unisys Code 302 Code 300.1, 1998.

[8] IEEE Std 610.12-1990. IEEE Standard Glossary of Software Engineering Terminology, IEEE Computer Society, 1990.

[9] URL: http://www.sqa.net/ iso9126.html

[10] J. Voas and K. W. Miller, Sematic Metrics for Software Testability, Journal of System and Software, Volume 20, pp: 207- 216, 1993.

[11] Shyam R. Chidamber, Chris F. Kemerer, A Metrics Suite for Object Oriented Design, IEEE Transactions on Software Engineering, Volume 20, Issue 6, pp: 476-492, June 1994.

[12] V. Gupta, K. K. Aggarwal and Y. Singh, A Fuzzy Approach for Integrated Measure of Object Oriented Software Testability, Journal of Computer Science, Volume 1, Issue 2, pp : 276-282, 2005.

[13] M. H. Tang, M. H. Kao and M. H. Chen, Empirical Study on Object Oriented Metrics, Proceedings of 6th International Software Metrics Symposium, pp: 242-249, November 1999.

[14] P. Greenwood, *et al.*, On the Impact of Aspectual Decompositions on Design Stability: An Empirical Study, Berlin, Germany, pp: 176-200, July 2007.

[15] M. Bruntink and A. V. Deursen, Predicting Class Testability using Object Oriented Software, Proceedings of 4th International Workshop on Source code Analysis and Manipulation, pp : 1-10, Sept. 2004.

[16] R. V. Binder, Design for Testability in Object oriented Systems, Communication of the ACM, Volume 37,Issue 9, pp : 87-101, Sept.1994.

[17] L. C. Braind, Y. Labiche and H. Sun, Investigating the Use of Analysis Contracts to Improve the Testability of Object Oriented Code, Software- practice and Experience, Volume 33, Issue 7, pp : 637-672, June 2003.

[18] B. W. N. Lo and H. Shi, A Preliminary Testability Model for Object Oriented Software, Proceedings of International Conference on Software Engineering: Education and Practice, pp: 330-337, 1998.

[19] URL: http://www.myreaders.info/07-fuzzy systems.pdf

[20] URL: http://www.mathworks.com/help/pdf-doc/fuzzy/fuzzy.pdf

Abbreviations

CBO	Coupling Between Objects
CK	Chidamber and Kemerer
DIT	Depth Inheritence Tree
HW	Health Watcher
LCOM	Lack of Cohesion of Methods
NOC	Number of Childern
OO	Object Oriented
OOS	Object Oriented System
OOSE	Object Oriented Software Engineering
RFC	Response for a Class
SE	Software Engineering
WMC	Weighted Methods per Class

List of Figures