Estimating Reusability of Software Components Using Fuzzy Logic

A Thesis

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

Master of Science

in

Mathematics and Computing

Submitted by

Vidhu Bhardwaj

(Registration No. 300803020)

Under the supervision of

Dr. Rajesh Kumar

School of Mathematics and Computer Applications,

Thapar University,

Patiala –147004 (PUNJAB)

INDIA

JULY 2010

Certificate

I hereby certify that the work is being presented in the thesis entitled "Estimating Reusability of Software Components Using Fuzzy Logic" in partial fulfillment 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.

Vidhu Bhardwaj

(Vidhu Bhardwaj)

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

13/07/10

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

Countersigned by:

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

Dr. R.K. Sharma
Dean of Academic Affairs
Thapar University, Patiala.

Acknowledgement

The key elements concentration, dedication, hard work and application are not the only essential factors for achieving the desired goals but also guidance, assistance and co-operation of people is necessary.

I would like to express my deep and sincere gratitude to my supervisor Dr. Rajesh Kumar, Associate Professor, School of Mathematics and Computer Applications (SMCA) for his able guidance, first hand experience, motivation 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 am grateful 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 Dr. A. K. Lal, Associate Professor, SMCA as P.G. Coordinator.

My thesis work would have incomplete without thanking my friends, who were always there in the hour of need.

I am very fortune to have unconditional support from my family. I thank my parents, who gave me the courage to get my education, supported me in all achievements throughout my life. Without their encouragement, this work would indeed have been very difficult for me to tackle.

Above all, I pay my reverence to the almighty GOD.

Vidhu Bhardwaj

(300803020)

Contents

Abstract

The last decade has shown that object-oriented concept by itself is not that powerful to cope with the rapidly changing requirements of ongoing applications. Component-based systems achieve flexibility by clearly separating the stable parts of systems (i.e. the components) from the specification of their composition. In order to realize the reuse of components effectively in Component Based Software Development, it is required to measure the reusability of components. However, due to the black-box nature of components, where the source code of these components are not available, it is difficult to use conventional metrics in Component based Development as these metrics require analysis of source codes. In this thesis, we adopt Fuzzy logic based approach to estimate the reusability of component. Several factors of reusability are taken into account.

Chapter

1

Introduction

The last decade has shown the object-oriented technology alone is not enough to cope with the rapidly changing requirements of present day applications. One of the reasons is that, although the Object-Oriented (O-O) methods encourage one to develop rich models that reflect the object of problem domain, this does not necessarily yield software architectures that can be easily adapted to changing requirements. Moreover, today's applications are large, complex and are not integrated. Although they come packaged with a wide range of features but most features can neither be removed, upgraded independently or replaced nor can be used in other applications. In particular, O-O methods do not typically lead to designs that make a clear separation between computational and compositional aspects [15].

An application must have some additional characteristics like robustness, usability, flexibility, simple installation, reusability, portability, interoperable, proper documentation etc. to fight with the advancement in the technology and rapidly changing requirements. To improve the business performance, it is necessary to use the latest technologies available.

Today Component Based Software Development (CBSD) is getting accepted in the company or an industry as a new effective development paradigm. It emphasizes the design and construction of software system using reusable components. CBSD is capable of reducing development cost and increasing the reliability of entire software system using components. The major advantages of CBSD are in-time and high quality solutions. Higher productivity,

flexibility and quality through reusability, replace-ability, efficient reusability and scalability are some additional benefits of CBSD [5].

Component Based Software Engineering (CBSE) is a paradigm that aims at constructing and designing systems using a pre-defined set of software components explicitly created for reuse. According to Clements [19], CBSE embodies the “the ‘buy, don’t build’ philosophy”. He also says about CBSE that “in the same way that early subroutines liberated the programmer from thinking about details, CBSE shifts the emphasis from programming to composing software systems”. The focus of CBSE is reusing whole software component, not objects. An important motivation for many organizations for adopting CBSE as their software development paradigm is to reduce development cost. One of the main contributions that CBSE has to this objective is the reuse of software components in multiple systems. In this way a software component is developed only once and can save out development effort multiple times.

1.1 Component Based Software Engineering

In Object-Oriented Programming (OOP), code is reused in the form of objects, and several mechanisms such as inheritance and polymorphism let the developers reuse these objects in several ways. The principle is the same with component-based software engineering (CBSE) also, but here the focus is on reusing whole software component, not just the objects. CBSE is a paradigm that aims at constructing and designing systems using a pre-defined set of software components explicitly created for reuse. It shifts the emphasis from programming to composing software systems.

There are many software component models available in the industry, some of these are *Microsoft’s* COM (Component Object Model), DCOM, .NET Framework, *Sun’s* Java Beans, EJB (Enterprise Java Beans), J2EE specification and *OMG’*s (Object Management Group) CORBA (Common Object Request Broker Architecture) specification and others [5].

CBSE uses Software Engineering principles to apply the same idea as OOP to the whole process of designing and constructing software systems. It focuses on *reusing* and *adapting* existing components, as opposed to just coding in a particular style. CBSE encourages the composition of software systems, as opposed to programming them.

Component-based development (CBD) is expected to have bright future and a tremendous scope for research. Present work in the thesis explores reusability feature for components and component-based systems and proposes new metrics for reusability of component. It uses Fuzzy Logic (FL) to design and evaluate reusability metrics.

1.2 Software Metrics

As the number of components available on the market increases, it is becoming more important to devise software metrics to quantify the various characteristics of components and their usage. Software metrics are intended to measure the software quality and performance characteristics quantitatively, encountered during the planning and execution of software development. These can serve as measures of software products for the purpose of comparison, cost estimation, fault prediction and forecasting.

Metrics can also be used in guiding decisions throughout the life cycle, determining whether software quality improvement initiatives are financially worth while [1].

A lot of research has been conducted on software metrics and their applications. Most of the metrics proposed in literature are based on the source code of the application.

However, these metrics cannot be applied on components and component-based systems (CBS) as the source code of the components is not available to application developers. Therefore, a different set of metrics is required to measure various aspects for CBS and their quality issues. The following section

discusses several metrics to measure various aspects, including complexity, reusability and others for legacy as well as for component-based systems [5].

1.3 Reusability

Reusability is the degree to which a component can be reused and reduces the software development cost by enabling less coding and more integration [2]. The reusability of assets is different in different contexts. However, there are some characteristics that generally contribute to the reusability of assets. Although many of these characteristics apply to assets in general, we focus in this section, we focus on components as assets. At a high level, we distinguish two aspects of reusability i.e. usability and usefulness [10].

Reusability = Usability + Usefulness

Usability is the degree to which an asset is ‘easy’ to use in the sense of the amount of effort that is needed to use an asset. Usability as such is independent of functionality of the component. Usefulness is the ‘frequency’ of suitability for use i.e. usefulness depends on the functionality, the generality and quality of a component [10].

1.4 Fuzzy Logic

Fuzzy logic is a methodology to solve problems which are too complex to be understood quantitatively. The fuzzy set theory introduced by Prof. Zadeh in 1965. Use of fuzzy sets in logical expression is known as Fuzzy Logic (FL) that has been the subject of important investigations. The central assertion underlying this approach is that entities in the real world simply do not fit into neat categories. For example, a project is not small, medium, or large. It could in fact be something in between; perhaps mostly a large project but also something likes a medium project. In the thesis, we proposed a Fuzzy Logic based approach for estimating reusability for component -based systems.

Chapter

2

Component Based Software Engineering (CBSE)

"CBSE is a process that aims to design and construct software systems using reusable software components". If one is familiar with Object Oriented programming (OOP), it can be useful to think of CBSE in a similar way. In OOP, code is reused in the form of objects. These objects are often contained in vast libraries of reusable code. Frameworks take the process even further, providing more robust and disciplined systems of reuse. By obtaining and reusing parts of systems which have already been 'tried and tested', we can exploit the principal advantages of OOP techniques over procedural programming techniques. They enable programmers to create modules1 that do not need to be changed when a new type of object is added. A programmer can simply create a new object that inherits many of its features from existing objects. This makes object-oriented programs easier to modify. In the same way, in CBSE, by reusing an existing component you cut out a lot of the hard work with establishing the usefulness and in testing that component. Although some testing will still be required [9,8].

CBSE should, in theory, allow software systems to be more easily assembled, and less costly to build. Although this cannot be guaranteed, the limited experience of adopting this strategy has shown it to be true. The software systems built using CBSE are not only simpler and cheaper, but usually turn out to be more robust, adaptable and updateable. CBSE allows use of predictable

architectural patterns and standard software architecture leading to a higher quality end result.

CBSE is an approach to software development that relies on reuse. It emerged from the failure of object-oriented development to support effective reuse. Single object classes are too detailed and specific. Components are more abstract than object classes and can be considered to be stand-alone service providers [13]

2.1 Component

A component is a reusable, self -contained piece of software with well-specified interface that is independent of any application. The very important point which has to be kept in mind, while developing a component is the reusability aspect, regardless of whether or not an organization can identify what the future requirements of the component will be. Components can be placed on any network node, depending on application needs and regardless on the type of particular network structure. An extra effort must be paid for the additional functionality of the component beyond the current application's need, to make the component more useful [13].

Components provide a service without regard to where the component is executing or its programming language, so:

- A component is an independent executable entity that can be made up of one or more executable objects.
- The component interface is published and all interactions are through the published interface.

Components can range in size from simple functions to entire application systems [13].

A *component* may be thought of as an independent module that provides information about what it does and how to use it through a public interface, while

hiding its inner workings. The interface identifies the component, its behaviors, and interaction mechanisms [21].

Today, components are seen as a step beyond objects. Like objects, components use contractually specified interfaces to implement the requirement of plug-compatibility with other components. But they go beyond objects by better addressing the requirement for replace-ability and by enabling the reuse of software parts that are larger-grained and at higher levels of abstraction. In contrast to objects, components can be large-grained as well as small-grained software parts. Large-grained components (enterprise components) encapsulate major chunks of an application's functionality in an independent, reusable, and easily replaceable unit. Because of their size, fewer enterprise components are needed to construct an application than objects. This solves the scaling problem with fine-grained components where integration and assembly become very tedious because of the number of parts needed to construct a large application [21].

Table 1: Examples of Components

- Order entry process (full application)
- Customer service screen
- Customer management component
- Validation component that interacts with multiple DB2 tables
- Workflow component
- Shipping component (order entry and shipment specification)
- Charge amount
- Business Associates
- Container type
- Unit of measure
- Event notification (infrastructure service)
- User interface controls
- Inter-application communication protocols
- Invoice component

- Order component
- Security
- General ledger
- Manufacturing work-in-process module
- Inventory tracking application

2.2 Characteristics of Components

The characteristics of a component are:

- It is a marketable entity. A component is a self-contained, shrink-wrapped, binary piece of software that one can typically purchase in the open market.
- It is not a complete application. A component can be combined with other components to form a complete application. It is designed to perform a limited set of tasks within an application domain.
- It can be used in unpredictable combinations. Like real-world objects, a component can be used in ways that were totally unanticipated by the original developer. Typically, components can be combined with other components of the same family – called suites using plug-and-play.
- It has a well-specified interface. Like a classical object, a component can only be manipulated through its interface. This is how the component exposes its function to the outside world. A component's interface is separate from its implementation. A component can be implemented using objects, procedural code, or by encapsulating existing code.
- It is an interoperable object. A component can be invoked as an object across address spaces, networks, languages, operating systems, and tools. It is a system-independent software entity.
- It is an extended object. Thus a component is a reusable, self-contained piece of software that is independent of any application.

- A software component is a reusable piece of code or software in binary form, which can be plugged into components from other vendors with relatively little efforts.
- A software component is a unit of packaging, distribution or delivery that provides services within a data integrity or encapsulation boundary [20].

2.3 Component Based Development

The Component-Based Development, also called development “with reuse”, deals with the applications construction. These applications are composed by reusing existing components. If needed, new components can be developed, or even acquired from third party. An important issue in CBSE is that it is more desirable to reuse an existing component than to develop a new one, even if the requirements are not entirely attended. This reduces both cost and time to market, but requires a searching mechanism to access all the available components. Thus, to be effective, a CBSE must provide ways for finding, connecting and adapting existing components. It must also support the addition of components that are developed or acquired during this phase.

Many information-based legacy systems contain similar or even identical things, which are developed from scratch again and again. From the scratch, development is more expensive and can take a long time to complete. Critical applications with strict time limits may loose the market due to the delay in the development process. This has led to the evolution of a new approach, called component -based development (CBD), which uses the concept of reusability in the application development. The high productivity is achieved by using standard components. CBD can be best described by the following two guiding principles:

- Reuse but do not reinvent;
- Assemble pre-built components rather than coding line by line.

Here components are viewed as families of routines that are constructed on the basis of well-defined principles so that these families fit together as building

blocks. In the last few years, CBD approach has become very popular. Component -based systems achieve flexibility by clearly separating the stable parts of the system (i.e. the components) from the specification of their composition [3].

2.4 Benefits of CBSE

The advent of component software is one of the most important new developments in the software industry since the introduction of high-level programming languages. Component software combines the advantages of custom software and standard software. It enables solutions that are better evolvable and more readily maintainable, can be extended over time, and can be modernized incrementally [21].

Business benefits of CBSE

Business benefits of component based software engineering are

Short Time to Market: Applications with components can be delivered synchronously with the business change cycle time

Adaptable: Componentized applications can be able to respond rapidly to change without forcing costly rewrites of the entire system.

Supporting Integration: There is often no time or justification to replace legacy applications. New functionality can be integrated with other packages, existing applications, and data sources.

Applicable: Componentized applications can align with the business. It is unacceptable to require the business to change to the application functionality.

Upgradeable: Improvements to an existing function can be possible without impact to other functions.

Technical benefits of CBSE

In the view of component users, the Technical benefits can be summarized as followings:

Power users will find it second nature to assemble their own personalized applications using off-the-shelf components. They will use scripts to tie the parts together and customize their behavior.

Small developers will find that components reduce expenses and lower the barriers to entry in the software market. They can create individual components with the knowledge that they will integrated smoothly with existing software created by larger development shops - they do not have to reinvent all the functions around them. They can get fine grained integration instead of today's 'band-aid' integration. In addition, they get faster time to market because the bulk of an application is already there.

Large Developers and System integrators will use component suites to create enterprise-wide client/server applications in record time. Typically, about 80% of the function they need will be available as off-the-shelf components. The remaining 20% is the value-added they provide. The client/server systems may be less complex to test because of the high reliability of the pre-tested components. The fact that many components are black-boxes reduces the overall complexity of the development process.

Desktop vendors will use components to assemble applications that target specific markets. Instead of selling monster suites at high prices, they will be able to provide their consumers with what they really need.

Chapter

3

Component Development for Reuse

The Component Based Development, also called development "with reuse", deals with the applications construction. These applications are composed by reusing existing components. If needed, new components can be developed, or even acquired from third party. An important issue in CBSE is that it is more desirable to reuse an existing component than to develop a new one, even if the requirements are not entirely attended. This reduces both cost and time to market, but requires a searching mechanism to access all the available components. Thus, to be effective, a CBSE must provide ways for finding, connecting and adapting existing components. It must also support the addition of components that are developed or acquired during this phase. There are three stages in this process, which are as follows;

- Component Qualification.
- Component Adaptation.
- Component Composition.

Components for reuse may be specially constructed by generalizing existing components.

Component reusability:

- Should reflect stable domain abstractions.
- Should hide state representation.
- Should be as independent as possible.
- Should publish exceptions through the component interface.

There is a trade-off between reusability and usability. The more general the interface, the greater the reusability but it is then more complex and hence less usable [13].

3.1 Reusable Component

Component based development and software reuse places new demands on software testing and quality assurance. This is especially true if the components are to be traded between organizations. A component will die or live by its quality. If the component does not possess the properties required for reuse, it will die its own death. Component should undergo strenuous tests, perhaps more than other software programs because errors will be introduced into every application that utilizes the components. If the project has very high reliability and availability requirements, the reputation of the developer is at stake [13].

A component is fit for reuse, if it provides the required behavior in the intended context. If we want high levels of reuse, then various contexts for the usage of the components should be considered. A software component may be used for many different applications, in different business and technical environments, by different developers using different methods and tools, for different users in different organizations [13].

Some of the *properties* that a reusable component must possess are as follows:

a. ***Well documented*:** Proper documentation assists a buyer to assess the credibility and validate the functional and quality properties of a component. As a means to qualify a component its documentation is a fundamental factor. In order to get the users assured that the quality of a component is in conformance with the quality of the product(s) the component will be used in thorough documentation is the only way. The information in component documentation is divided into four categories –

general information, detailed information, acceptance test, support and additional information (such as unsolved bugs and problems and their severity and state).

b. ***Conformity to standards*:** Standardization is a good approach for developing a common component market. A component model specifies the standards and conventions imposed on developers of components. Compliance with a component model is one of the properties that distinguish a component from other forms of packaged software. Standards are beneficial in defining common interfaces and infrastructure for interoperation between components, which further helps in reducing the chances of mismatch that inhibit the composition of components.

c. ***Availability of related information***: Several requirements and testing activities can decrease potential problems associated with component reuse and thereby improve the resulting software's quality. A good component has with it associated several related artifacts, which are appropriately updated when the component is modified. The artifacts mostly include – the software specification, the test suite, links of the software specifications with the test cases in the test suite. When parts of the specification change, it becomes easier to identify those parts of the test suite that require enhancement and need to be rerun.

d. ***Vendor support***: a component will have some standing in the market if it is supported by an economically stable vendor (component supplier). There is no fear of the vendor going out of business. Or if something of this nature happens i.e. the vendor no longer supports the component, the legal agreement between the component user and component supplier forces the component supplier to provide the source code of the component to the component user.

e. ***Certified*:** the long term success of software component technology depends upon availability of high quality components and trust on the part of consumers that the components that they buy are of high quality. The component users want to know that a component will function as

advertised. So the component is available along with the valid certificates issued by a component certifier.

The requirements that a component should meet, are defined by the architecture in which the component is intended to be used. The requirements specification standard classifies requirements to functional and non-functional (quality) requirements. It is the responsibility of an architect to define the functional and quality requirements of a component [13].

3.2 Reusability of Software Component

Software programming is a highly specialized and intellect -oriented task, mainly due to the complexity involved in the process. Nowadays, this complexity is increasing to levels, in which reuse of previous software designs are helpful to reduce the efforts and total development time. The main idea of software reuse is to use previous software components/artifacts to create new software systems. Thus, software reuse is software design, where already developed components are the building blocks for the creation of new systems. The main objective of software reuse is to minimize repetition of work, development time, cost and efforts and increase reliability of the systems. It also improves the reusability and portability of the system [3].

In Component based development of software, there are two broad reuse developments. One is development of systems with reuse and another is development of components for reuse. In first case, application is developed by reusing several already built-in components. These components are already tested thoroughly, which will enhance the quality of the end product and will save time and cost. Another approach is the development of components for reuse. Here, components are developed keeping in mind the high reusability. These components need to be compatible for wide range of applications developed in variety of languages for different platforms. Present chapter emphasizes on the first aspect of reuse and proposes a methodology to estimate the reusability of the target component before using it in the final system. It will help application

developers to compare and chose the highly reusable component among others and will eventually lead to the application development with low maintenance efforts [3].

3.3 Reusability Metrics

Reusability can measure the degree of features that are reused in building applications. There is a number of metrics available for measuring the reusability for object-oriented systems. These metrics focus on the object structure, which reflects on each individual entity such as methods and classes, and on the external attributes that measures the interaction among entities such as coupling and inheritance. But there are some difficulties in applying existing object oriented metrics into the component development and CBSD. Object oriented metrics cannot be used to measure the component's quality. The reasons are:

a. *Measurement unit is different*. Object oriented metrics only focus on objects or classes. Component consists of one or more classes as well as one or more interfaces. Existing object-oriented metrics do not consider component itself or component's interfaces on measuring complexity, cohesion or coupling and so on. Therefore it is required new metrics that measure complexity of components itself.

b. *Measurement factor is insufficient*. Because object oriented applications are developed with only classes, most of the object-oriented metrics measure the complexity or reusability by considering classes, methods and depth of class hierarchy. However, considering these factors are not sufficient to measure the complexity or reusability of component because components have more much information such as interfaces, interface methods and so on.

c. *Existing object oriented metrics do not consider customizability of classes or objects*. Customizability of components is very important in CBSD because component's customizability effects on reusability of components in CBSD. Most of the traditional metrics are based on source code (LOC) or similar size counts, defects counts and effort figures. For O-O Development also, reuse is assumed

to be a very effective strategy to build high-quality software. All these approaches use the size factor to measure the empirical values of the quality attributes of the software. However in CBD, the metrics are different than the conventional metrics. Components are termed as black box entities, for which size is not known so alternative measures have to be used to measure the quality of the software [3].

Poulin [16] presents a set of metrics used by IBM to estimate the efforts saved by reuse. The study suggests the potential benefits against the expenditures of time and resources required to identify and integrate reusable software into a product. Study assumes the cost as the set of data elements like Shipped Source Instructions (SSI), Changed Source Instructions (CSI), Reused source Instructions (RSI) etc.

Reuse Percentage measures how much of the product can be attributed to reuse and is given as

*Product Reuse Percentage = (RSI / (RSI + SSI)) * 100%*

Paper proposes several other reusability metrics in terms of cost and productivity like Reuse cost avoidance, Reuse value added and Additional development cost, which can be used significantly for business applications.

Cho et al [7] proposes a set of metrics for measuring various aspects of software components like complexity, customizability and reusability. The work considers two approaches to measure the reusability of a component. The first is a metric that measures how a component has reusability and may be used at design phase in a component development process. This metric, Component Reusability (CR) is calculated by dividing sum of interface methods providing commonality functions in a domain to the sum of total interface methods. The second approach is a metric called Component Reusability level (CRL) to measure particular component's reuse level per application in a component based software development. This metric is again divided into two sub-metrics. First is CRLLOC, which is measured by using lines of code, and is expressed as percentage as given as

CRL LOC (C) = (Reuse (C) / Size (C)) *100%

The second sub-metric is CRL Function, which is measured by dividing functionality that a component supports into required functionality in an application. This metric gives an indication of higher reusability if a large number of functions used in a component. However, the proposed metrics are based on lines of codes and can only be used at design time for components.

Sharma et al. [3] have considered four factors of reusability namely customizability, interface complexity, understandability and portability of component and used Fuzzy Logic based approach to assess reusability of software components.

Present thesis works on the reusability metrics for software components by considering five factors customizability, interface complexity, understandability, commonality and portability and applies FL based approach for estimating reusability.

3.4 Benefits of Reuse

Some of the benefits of reusability of a software component are as follows:

Increased reliability - Components exercised in working systems.

Reduced process risk - Less uncertainty in development costs

Effective use of specialists - Reuse components instead of people.

Standards compliance - Embed standards in reusable components.

Accelerated development - Avoid original development and hence speed-up production [13].

Not only there are tremendous benefits to be gained from reuse, but also there are tremendous opportunities to employ reuse in software projects. Components are the software industry's latest attempt to capitalize on this similarity. Backed by advances in technologies and tools, components offer the best chance yet to achieve a significant level of reuse in industrial-strength application development projects [21].

Chapter

4

Estimating Reusability of Component

In case of component-based development, software reuse refers to the utilization of a software component with in a product, where the original motivation for constructing this component was not known. Here, reuse is seen as black-box reuse, where the application developer sees the interface, not the implementation of the component. The interface contains public methods, user documentation, requirements and restrictions of the component. If there is a change in the code of a black-box component, compiling and linking the component would propagate the change to the applications that reuse the component. As the users of the component trust its interface, changes should not affect the logical behavior of the component [3].

As the number of components available in the market increases, it is becoming more important to devise software metrics to quantify the various characteristics of components. Among several quality characteristics, the reusability is particularly important when reusing components. Reusability can measure the degree of features that are reused in building applications [3]. In the present study, we have taken into account, the understanding of the components as well, along with other important factors required for estimating the reusability of software components.

4.1 Proposed Methodology for Estimating The Reusability for Components

In this chapter, we proposed a Fuzzy Logic based approach for estimating reusability for component based systems.

It is often impossible to estimate software quality attributes directly. For example, attributes (say, reusability, etc.) are affected by many different factors, and there is no straightforward method to measure them. To estimate reusability of CBS, one needs to establish a relationship of the factors with reusability to achieve the desired goal. Emphasis in the present work is to estimate reusability, if the changes require customization/replacement of the component or integration code has to be modified. Following factors have been identified, which will influence reusability of CBS:

a. Customizability
b. Interface Complexity
c. Understandability
d. Commonality
e. Portability

a. ***Customizability***

Customizability is defined as the ability to modify a component as per the application requirement. Better reusability can be achieved if customizability is better, thus help in maintaining the component in the later phases [3]. It may be measured on the basis of writable properties available in the component.

The following formula is used to evaluate this metric [4]:

$$\text{Customizability} = \frac{\text{No. of Set Methods}}{\text{Total number of Properties}}.$$

By using this metric, one can measure that how much an interface method can be customized. Therefore, it may be used to measure the reusability of the component. Customizability of a component may vary from 0 to 1.

b. ***Interface Complexity***

As components are black box in nature, so we are unable to get the source code of these components. Application may interact with these components only through their well - defined interfaces. Interface acts as a main source for understanding, use and implementation and finally maintenance for the component. Therefore, the complexity of these interfaces plays a vital role while measuring the overall complexity of the component. Complex interfaces will lead to the high efforts for understanding and customizing the components. Therefore for better reusability, interface complexity should be as low as possible [3]. The most widely used complexity metric is Cyclomatic Complexity proposed by McCabe (1976). This metric is based on the program graph and is defined as:

$$V(G) = e - n + 2p$$

Where ***e*** is the number of edges, *n* is the number of nodes in the graph and p is the number of connected components. Author proposed that *V(G)* can be used as a measure of procedure complexity of the program.

c. ***Understandability***

Since the source code of the component is not available to the application developer, we can understand the component through the document only. Documentation provides the ease with which a user can learn to operate, prepare inputs for, and interpret outputs of a system or component.

Here, the term documentation of component refers to component manuals, demos, help system, and marketing information. Functional elements may be referred as the interfaces, operations, or events that a component may support or require from other components to achieve its functionality, i.e., to implement its services. A good quality document must include functional description, installation details, system administrator's guide, system reference manuals etc. It may also require non -functional details, like performance, security issues, and previous maintenance activities, if any. It will help in understanding the component and can be reused easily in the final system and also help in implementing the maintenance activities with less effort and in an effective

manner [3]. For present work, we categorize documentation quality into very low, low, medium, high and very high categories.

d. ***Commonality***

Commonality for both functionality and reusability factors measures how well the functionality of components matches to the functionality of system and its standard domain model. If a component provides a superior functionality but does not meet the commonness of the domain, its reusability among organizations will be limited.

e. ***Portability***

It is the ability of a component to be transferred from one environment to another with slight modification, if required. It is typically concerned with reuse of component on new platforms. The component should be easily and quickly portable to specified new environments if and when necessary, with minimized porting efforts and schedules. For better reusability, component should be highly portable, i.e. component should be supported by many operating systems [3].

The goal of our work is to develop a tool for estimating reusability of the software component. We consider that reusability is a measure of factors mentioned above. The values of these individual factors can be measured by using the appropriate metrics. Customizability metric is the ratio of writable properties to the total number of properties. Documentation and portability can be classified from very low to very high categories.

4.2 Fuzzy Logic

Fuzzy logic is a methodology to solve problems which are too complex to be understood quantitatively, based on fuzzy set theory, which introduced by Prof. Zadeh in 1965 [18]. Use of fuzzy sets in logical expression is known as FL that has been the subject of important investigations. In the beginning of the nineties, FL was firmly grounded in terms of its theoretical foundations and used

in various fields. The central assertion underlying this approach is that entities in the real world simply do not fit into neat categories. For example, a project is not small, medium, or large. It could in fact be something in between; perhaps mostly a large project but also something likes a medium project. This can be represented as a degree of belonging to a particular linguistic category. Fuzzy sets can be effectively used to represent linguistic values such as low, young, and complex [21].

A fuzzy set can be defined mathematically by assigning to each possible individual in the universe of discourse a value representing its grade of membership in the fuzzy set to a greater or lesser degree as indicated by a larger or smaller membership grade. Major advantage of this approach is that it is less dependent on historical data. A FL system which accepts imprecise data and vague statements such as low, medium, high and provides decisions [21] as shown in figure 1 below.

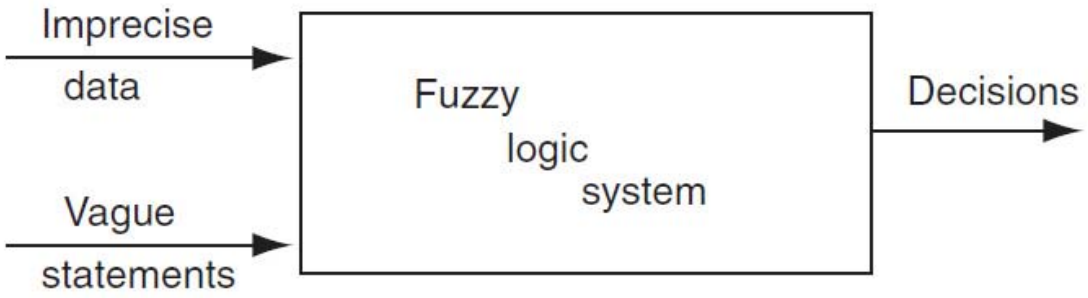

Fig 1: Fuzzy Logic System

4.3 Implementation of Fuzzy System

Implementing a fuzzy system requires that the different categories of the different inputs be represented by fuzzy sets which, in turn, is represented by membership functions. The domain of membership function is fixed, usually the

set of real numbers, and whose range is the span of positive numbers in the closed interval [0, 1]. There are total 11 membership functions available in MatLab. We considered Triangular Membership Functions (TMF) for our problem, because of its simplicity and heavy use by researchers for prediction models [18]. It is a three-point function, defined by minimum α, maximum β and modal value m i.e. TMF (α, m, β), where ($\alpha \leq m \leq \beta$). This process is known as fuzzification. These membership functions are then processed in fuzzy domain by inference engine based on knowledge base (rule base and data base) supplied by domain experts and finally the process of converting back fuzzy numbers into single numerical values is called defuzzification [17].

4.4 Fuzzy Model

We propose that reusability of component-based system is a measure of five factors mentioned above. These combined factors can be used to measure the reusability, as it cannot be measured directly. The proposed FL based model considers all five factors as inputs and provides a crisp value of reusability using the Rule Base. All inputs can be classified into fuzzy sets *viz*. Low, Medium and High. The output reusability is classified as Very High, High, Medium, Low and Very Low. All possible combinations (3^5 i.e. 243) of inputs are considered to design the rule base. Each rule corresponds to one of the five outputs based on the expert opinions. Some of the proposed rules are shown as:

- If Customizability of components is Low, Interaction Complexity among component is High, Understandability is Low, Commonality is Low and Portability is Low then it is very difficult to maintain the system i.e. Reusability will be Very Low.
- If Customizability of components is Low, Interaction Complexity among component is High, Understandability is Low, Commonality is Medium and Portability is Medium then Reusability will be Low.

- If Customizability of components is Medium, Interaction Complexity among component is High, Understandability is Medium, Commonality is Medium and Portability is Medium then Reusability will be Medium.
- If Customizability of components is Medium, Interaction Complexity among component is Medium, Understandability is Medium, Commonality is Medium and Portability is Medium then Reusability will be Medium.
- If Customizability of components is Medium, Interaction Complexity among component is Low, Understandability is High, Commonality is Medium and Portability is Medium then Reusability will be High.
- If Customizability of components is High, Interaction Complexity among component is Low, Understandability is High, Commonality is Medium and Portability is Medium then Reusability will be High.

All 243 rules are inserted into the proposed model and a rule base is created. Depending on a particular set of inputs, a rule is fired. Using the rule viewer, output i.e. reusability is observed for a particular set of inputs using the MatLab Fuzzy tool box.

Table 2, 3 and 4 show the system configuration and values of various parameters set for inputs, outputs for fuzzification process.

Table 2: System in MatLab

Name	'reusability'
Type	'mamdani'
Version	2.0
NumInputs	5
NumOutputs	1
NumRules	243
AndMethod	'min'
OrMethod	'max'
ImpMethod	'min'
AggMethod	'max'
DefuzzMethod	'centroid'

Table 3: Parameter values for Inputs

Input 1	'Customizability'
Range	[0 1]
NumMFs	3
MF1	'Low':'trimf',[0 0.2 0.35]
MF2	'medium':'trimf',[0.3 0.5 0.68]
MF3	'high':'trimf',[0.65 0.8 1 .0]
Input 2	'Intercation_Complexity'
Range	[0 1]
NumMFs	3
MF1	'low':'trimf',[0 0.2 0.35]
MF2	'medium':'trimf',[0.32 0.5 0.68]
MF3	'high':'trimf',[0.62 0.8 1.0]
Input 3	'Understandability'
Range	[0 1]
NumMFs	3
MF1	'low':'trimf',[0 0.16 0.35]
MF2	'medium':'trimf',[0.3 0.5 0.68]
MF3	'high':'trimf',[0.62 0.85 1.0]
Input 4	'Commonality'
Range	[0 1]
NumMFs	3
MF1	'low':'trimf',[0 0.16 0.35]
MF2	'medium':'trimf',[0.3 0.53 0.68]
MF3	'high':'trimf',[0.63 0.8 1 .0]
Input 5	'Portability'
Range	[0 1]
NumMFs	3
MF1	'low':'trimf',[0 0.15 0.35]
MF2	'medium':'trimf',[0.32 0.55 0.66]
MF3	'high':'trimf',[0.63 0.76 1.0]

Table 4: Parameter values for Output

Output	'ReusabilityOfComponent'
Range	[0 1]
NumMFs	5
MF1	'Very_Low':'trimf',[0 0.1 0.21]
MF2	'Low':'trimf',[0.19 0.3 0.42]
MF3	'Medium':'trimf',[0.38 0.5 0.62]
MF4	'High':'trimf',[0.59 0.7 0.81]
MF5	'Very_High':'trimf',[0.78 0.9 1.0]

Fuzzification of inputs and outputs in fuzzy logic tool in MatLab software is shown in figure 2.

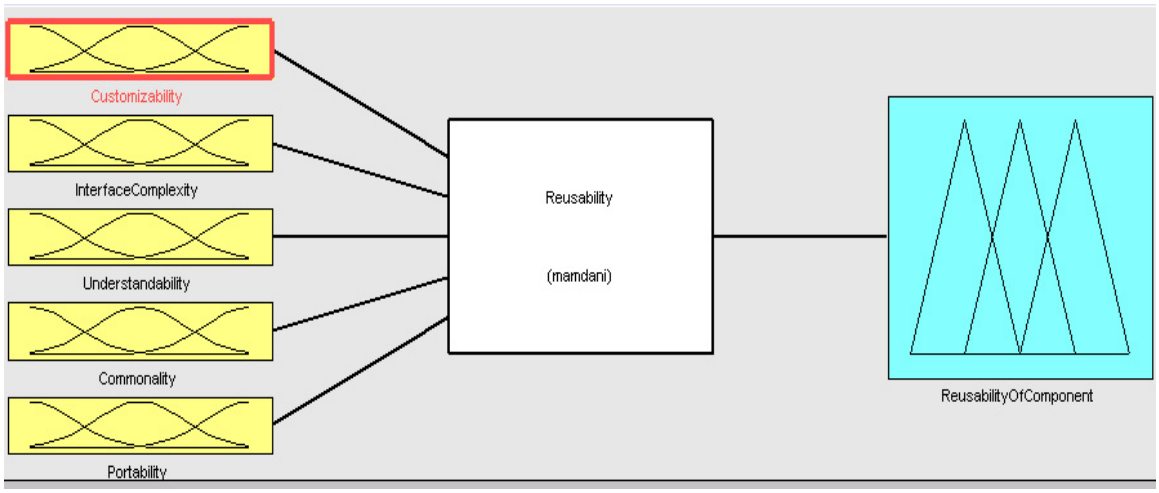

Fig 2: Fuzzification

Membership function for Customizability as input 1 in fuzzy logic tool in MatLab is shown below in figure 3.

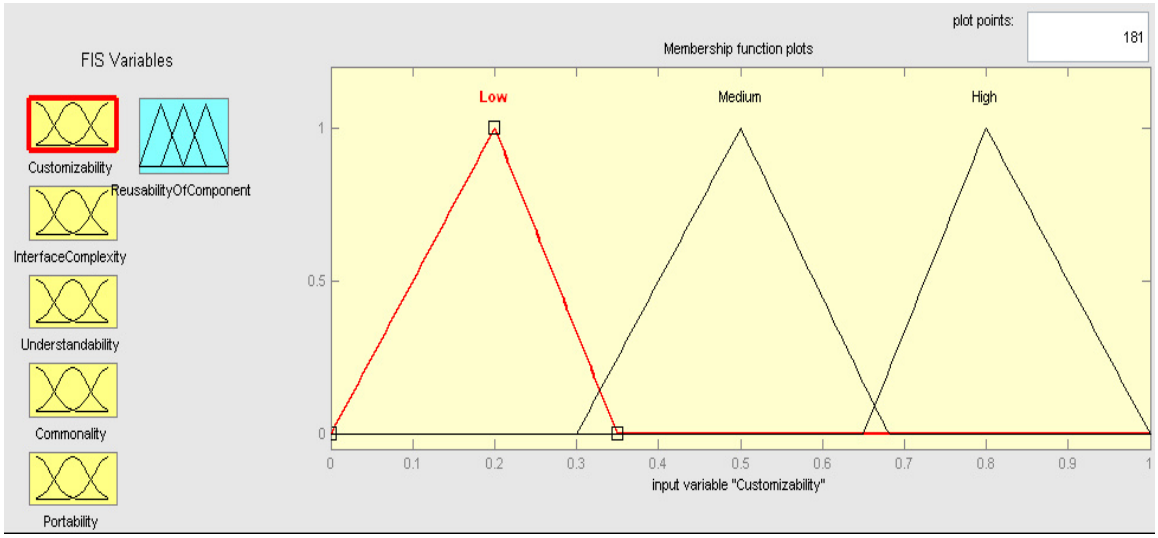

Fig 3: Membership Function.

After inserting the rules in purposed model, a rule base is created. So, using a rule viewer shown in figure 4, Reusability is observed by taking all five values for input factors.

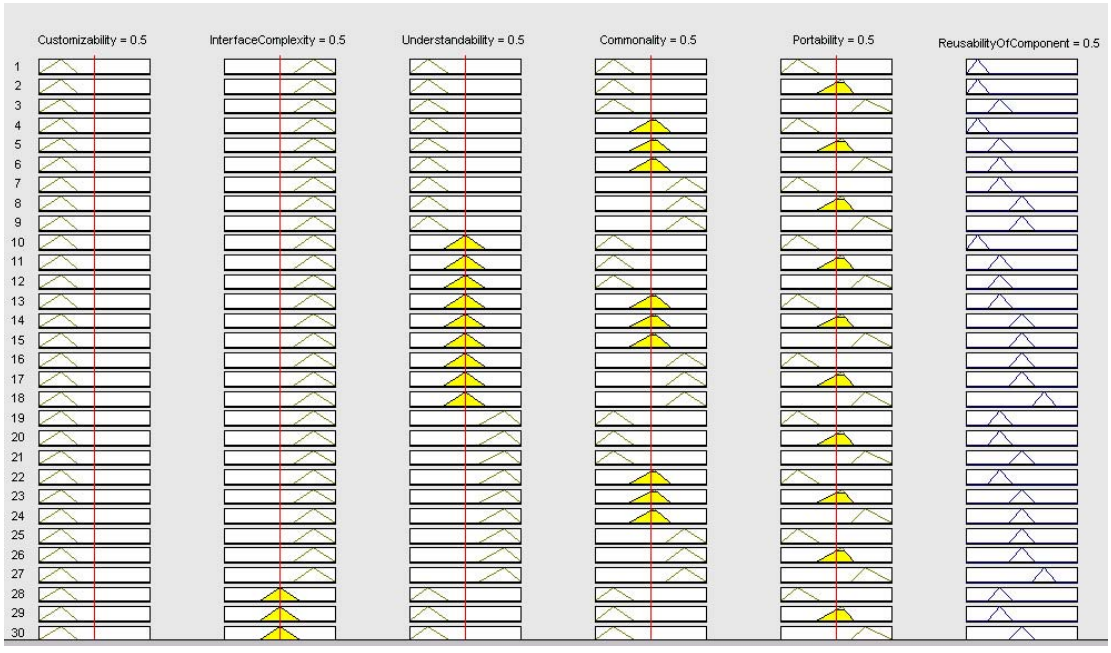

Fig 4: Rule Viewer

By using rule viewer, the output i.e. Reusability of a Component can be observed by taking different sets of inputs by dragging the central red line of inputs accordingly. One of the changes in above rule viewer is shown in figure 5, which reflects the change in output.

That is, for set of inputs [0.775, 0.215, 0.775, 0.765, 0.785], the output is 0.893.

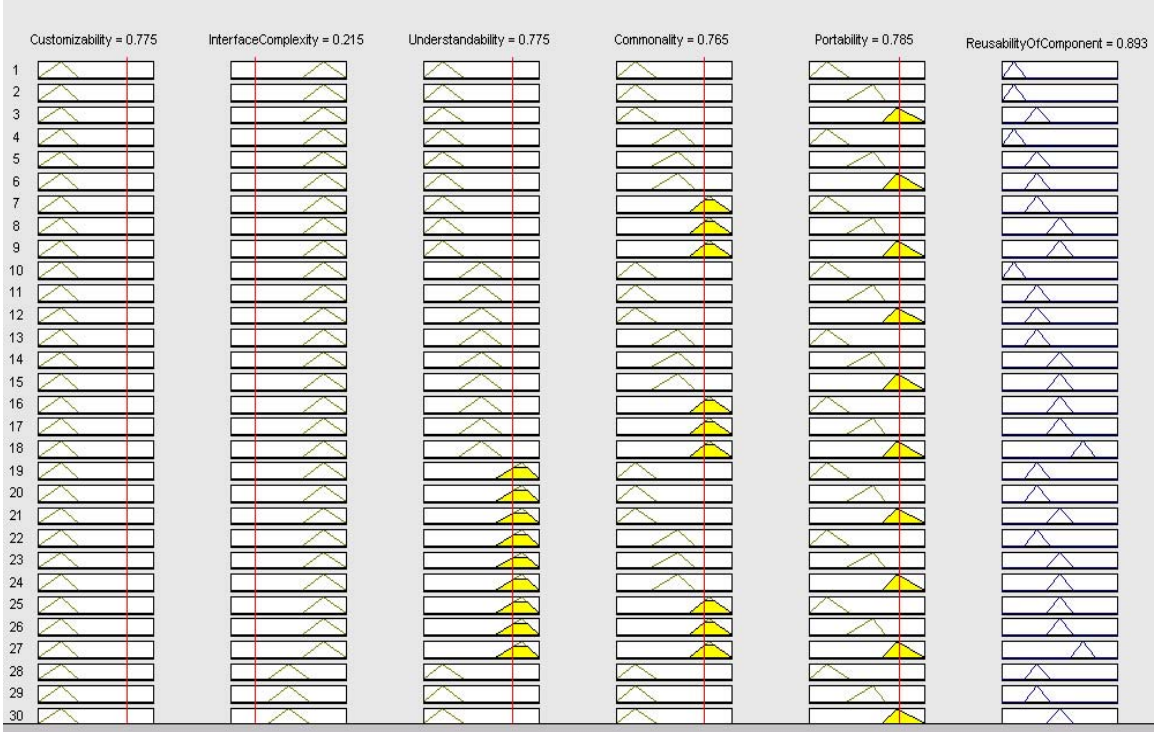

Fig 5: Rule viewer with changed values.

This shows that for better reusability of software component, its customizability, understandability, commonality and portability should be high whereas interface complexity should be low.

Chapter

5

Conclusion

Component-based software engineering (CBSE) (also known as component-based development (CBD)) is a branch of software engineering which emphasizes the separation of concerns in respect of the wide-ranging functionality available throughout a given software system. This practice aims to bring about an equally wide-ranging degree of benefits in both the short-term and the long-term for the software itself and for organizations that sponsor such software.

Software reuse is the process of creating software systems from existing software rather than building software systems from scratch. Reusable software components are designed to apply the power and benefit of reusable, interchangeable parts from other industries to the field of software construction. Reusable components add standardized interfaces and object introspection mechanisms to widgets allowing builder tools to query components about their properties and behavior. Software components need not be visible in a running application; they only need to be visible when the application is constructed.

Reusability metrics can measure the degree of features that are reused in building applications. Even if there are some metrics defined for the reusability of object-oriented software (OOS), they cannot be used for CBSD because these metrics require analysis of source code. Building software systems with reusable components bring many advantages to Organizations. Reusability may have several direct or indirect factors like cost, efforts, and time. It may also have the issues like whether reusability is for the entire component or only for a selected

service provided by that component. Paper discusses various aspects of reusability for Component-Based systems.

Reusability is the most important criteria for selecting a component for component-based systems. A highly reusable component will help in better understanding and low maintenance efforts for the application. Therefore, it is necessary to estimate the reusability of the component, before integrating it into the system. Present thesis adopts Fuzzy logic based approach to estimate the reusability of component. It also proposes to improve the Reusability factors in *Sharma et al*. [3] work by adding Commonality as a new factor in it. Fuzzy Logic based approach has several advantages over other methods including Neural Network and others. One major advantage is that it may also work without the data. So, Fuzzy logic will result in better understanding the reuse task. It considers factors like customizability, interaction complexity, understandability, commonality and portability as inputs while reusability of component is considered as output. In total 243 fuzzy sets have been designed and are represented by membership functions (Low, Medium and High). Output is represented by Very Low, Low, Medium, High and Very High. Defuzzification process produces the value of reusability corresponding to input sets. Results show that proposed model can be used to predict the reusability of CBS, which will help in estimating development efforts and quality for the application.

References

[1]. A. Gafoo, Sedigh-Ali and R. A. Paul: "*Metrics-Guided Quality Management for Component-Based Software Systems*", Proceedings of 25th Annual International Computer Software and Applications Conference (COMPSAC '01), 20001.

[2]. A. J. A. Wang, "*Reuse Metrics and Assessment in Component-based Development"*, Proceedings of 6th IASTED International Conference on Software Engineering and Applications, 2002.

[3]. A. Sharma: "*Design and Analysis of Metrics for Component-Based Software Systems*" Ph.D. thesis, Thapar University, Patiala, 2009.

[4]. A. Sharma, P. S. Grover, R. Kumar: "*Investigation of Reusability, Complexity and Customizability Metrics for Component-Based Systems*", ICFAI Journal of Information Technology, 2006.

[5]. A. Sharma, R. Kumar, and P. S. Grover: "*A Critical Survey of Reusability Aspects for Component-Based Systems*", World Academy of Science, Engineering and Technology, Vol. 19, pp: 411-415, 2007.

[6]. B. Li, "*Software Reuse*", 1998, available at http://sern.ucalgary.ca/courses /seng/693/ W98/ lib/reuse.htm

[7]. E. S. Cho et al.: "*Component Metrics to Measure Component Quality*", Proceedings of the eighths Asia-Pacific Software Engineering Conference, 2001.

[8]. F. Siddiqui: "*CBSE: A look at reusable software components*", 1999, available at http://www.smb.uklinux.net/reusability

[9]. G. Zhang: "*Component-Based Software Engineering*", Thesis, 2000.

[10]. H. Mili, F. Mili, A. Mili, "*Reusing Software: Issues and Research" Directions*, IEEE Transaction on Software Engineering, Vol. 21, Issue 6, pp: 528-561, 1995.

[11]. H. Washizaki, H. Yamamoto "*Metrics Suits for Measuring Reusability Of Software Components*", Proceedings of 9th International Symposium on Software Metric 2003.

[12]. http://www.mathworks.com/access/helpdesk/help/toolbox/nnet/purelin.html

[13]. Ian Sommerville: "*Software Engineering*", 7th edition, 2004.

[14]. IEEE 1620, "*IEEE Standard Glossary of Software Engineering Terminology*", the Institute of Electrical and Electronics Engineers, Inc. 1990.

[15]. J. G. Schneider: "*Component Scripts and Glue: A Conceptual framework for software composition*" Ph.D. thesis, Institute für Informatik (IAM), Universität Bern, Berne, Switzerland, 2003.

[16]. J. Poulin, J. Caruso and D. Hancock: "*The Business Case for Software Reuse* ", IBM Systems Journal, pp: 567-594, 1993.

[17]. K. K. Aggarwal, Y. Singh, P. Chandra, M. Puri, "*Measurement of Software Maintainability Using a Fuzzy Model"*, Journal of Computer Sciences, Vol. 1, Issue 4, pp: 538-542, 2005.

[18]. L. A. Zadeh: "*From Computing with numbers to computing with words – from manipulation of measurements to manipulation of perceptions*", International Journal of Applied Mathematics and Computer Science, Vol.12, Issue 3, pp: 307-324, 2000.

[19]. M. Sparling: "*Lessons Learned through Six Years of Component Based Development*", Communications of the ACM, 2003.

[20]. Microsoft Corporation. Definition of the term component; http://www.msdn.microsoft.com/repository/OIM/resdkdefinitionofthetermcomponent.asp

[21]. N. J. Piscataway, IEEE 1517, *"Introduction to IEEE Std. 1517—Software Reuse Processes",* IEEE, 1999.

[22]. N. S. Gill: "*Importance of Software Component Characterization for Better Software Reusability*", ACM SIGSOFT SEN, Vol. 31, Issue 1, pp: 1-3, 2006.

[23]. N. S. Gill, P. S. Grover, "*Component-Based Measurement: Few Useful Guidelines*", ACM SIGSOFT Software Engineering Notes, Vol. 28, Issue 6, pp: 1-4, 2003.

[24]. R. S. Pressman, "*Software Engineering: A Practitioner's Approach*", 6th Edition, McGraw Hill Book Co. 2005.

[25]. Siddiqui: "*CBSE: A look at reusable software components*", available at: http://www.smb.uklinux.net/reusability .

[26]. S. N. Sivanandam, S. Sumathi and S. N. Deepa: "*Introduction to fuzzy logic using MATLAB*", Springer, 2007.

[27]. T. H. George, T. William: "*Component Based Software Engineering*", Addison Wesley Longman, 2001.

Abbreviations

CBD	Component Based Development
CBS	Component Based System
CBSD	Component Based Software Development
CBSE	Component Based Software Engineering
CR	Component Reusability
CRL	Component Reusability Level
CSI	Changed Source Instruction
FL	Fuzzy Logic
O-O	Object-Oriented
OOP	Object Oriented Programming
OOS	Object Oriented System
RSI	Reused Source Instruction
SSI	Shipped Source Instruction
TMF	Triangular Membership Function

List of Tables

List of Figures