Enhancing Object Oriented Coupling Metrics w.r.t. Connectivity Patterns

Thesis submitted in partial fulfillment of the requirements for the award of degree of

Master of Engineering

in

Software Engineering

Submitted By

Sangeeta

(801231025)

Under the supervision of

Ms. Ashima Singh

Assistant Professor, CSED

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

THAPAR UNIVERSITY

PATIALA – 147004

JUNE 2014

Certificate

I hereby certify that the work which is being presented in the thesis entitled, **"Enhancing Object Oriented Coupling Metrics w.r.t. Connectivity Patterns"**, in partial fulfillment of the requirements for the award of degree of Master of Engineering in *Software Engineering* submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of Ms. Ashima Singh and refers other researcher's work which are duly listed in the reference section.

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

(Sangeeta)

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

(Ms. Ashima Singh)
Assistant Professor
Computer Science and Engineering Department
Thapar University
Patiala

Countersigned by

(Dr. Deepak Garg)

Head
Computer Science and Engineering Department
Thapar University, Patiala

(Dr. S. K. Mohapatra)

Dean (Academics 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.

First and foremost, I would like to express my deep and sincere gratitude to my supervisor Ms. Ashima Singh, Assistant Professor, Computer Science and Engineering Department, Thapar University, Patiala. But no volume of words is enough to express my gratitude towards my guide. Her wide knowledge and logical way of thinking have been a great value for me. Her understanding and personal guidance have provided a strong basis for my thesis. Without her mentoring this thesis would never have been realized.

I owe my sincere thanks to Dr. Deepak Garg, Professor and Head of Computer Science and Engineering Department, Thapar University, Patiala, for motivation and providing facilities.

I wish to express my sincere thanks to all the staff members and my friends who always supported and encouraged me. I was very fortunate to have an unconditional support from my family. I thank my family who gave me courage to get my education, supported me in all achievements throughout my life. Last but not the least; I would like to thank God for giving me inner peace and strength.

Sangeeta

Abstract

Several Object Oriented metrics have been proposed in the literature. These metrics aim to measure the relationship between different classes and their methods. There are different ways in which classes are connected. Different metrics use different models to represent the relationship or connectivity pattern between the classes. In some cases, the Object Oriented metrics obtain the same value of coupling for different components having same models (i.e. same number of classes in a component) but different connectivity patterns. This leads incorrectly considering the components to be same in terms of coupling, even though their relationship or connectivity patterns clearly indicate that the degree of coupling are different. We refer this problem as Inability to Differentiate Anomaly (IDA). In this thesis we list and discuss Object- Oriented metric like Coupling between Objects (CBO), Response for Class (RFC) and Depth Inheritance Tree (DIT) in which the Inability to Differentiate Anomaly exists. We empirically study the frequent occurrence of IDA problem when the considered metrics are applied to different components with same class model. Finally, we propose a metric i.e. Coupling based on Strength Specification Metric (CSSM) which gives the distinct coupling value for components with same number of classes but different connectivity patterns. We compared and contrasted CBO and $CSSM_{CBO,}$ RFC and $CSSM_{RFC,}$ DIT and $CSSM_{DIT}$ and quoted the differences very clearly. And finally we give normalized CSSM which lie between the ranges 0 to 1 which classifies the components into three categories i.e. complex, medium and low. Metrics based on the strength of coupling parameters is more desirable because it exhibits a lower chance of incorrectly considering components to be equally coupled when they have different connectivity patterns.

Table of Content

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview

Over a few decades, software systems are becoming more and more complex than before. It may result in many problems if we use traditional software development approach in the new situation. There are different problems like failure to meet quality, budget and deadline requirements. It would never be possible to overcome such kind of problems by developing the software products each time from scratch. CBSD is using the reusable components for constructing the software systems, which is becoming an effective and popular approach for software development [1]. Component based system provides different advantages like reduced development cost, time, efforts and increase quality. Large amount of standard components for a variety of functionalities must be available to make component-based development feasible. Measuring and controlling the software complexity is important phase of every software development. Component Based Software system complexity is dependent on the complexity of the already existing components. Coupling is defined as the strength of association which is developed by building a connection between components. We consider different object oriented coupling metrics which helps in measuring the coupling complexity of components. These metrics provides an idea about which component will have less complexity and which component will be easy to integrate with other component. The low level coupling components are more preferable. Based on the different aspects many researchers have proposed various types of object oriented coupling metrics like Coupling between Object (CBO), Response for a Class (RFC), Depth Inheritance Tree (DIT), etc, but these metrics does not consider the internal structure of the component (i.e. relation or connection among classes in a component) due to which they give ambiguous results. In this project we overcome the problem of ambiguous results of the existing coupling metrics like CBO, RFC and DIT (i.e. which gives the same complexity value for different components with same model) and give a unique result (i.e. which gives the different complexity value for different components with same model). We get the unique results of the components using the different connectivity patterns (i.e. UML relationships).

1.2 What is Component Based Software Engineering

For the development of software system, Component Based Software Engineering (CBSE) is a modern approach. In CBSE the software system are developed by integrating existing components. By using reusable software components CBSE focuses on the construction and design of software system [2] [7]. The main focus of CBSE is to build a software system by using the existing software components. It reduces the cost and time development of the software system. In controlling and measuring the software complexity CBSE provides the central problem. From the failure of OO development to support the effective reuse the component based software engineering approach has been emerged. Classes with single objects are too specifics and detailed. The components are considered to be stand alone service provider and they are more abstract than object classes. Due to the extensive use of components, the component based software engineering is different from traditional waterfall approach [1]. In CBSE the special life cycle phases are the component evaluation and component selection. But in CBSE many efforts are required in verification and testing phases.

1.3 Software Complexity Metrics

The main objective of software development is controlling and measuring the software complexity. There are many problem encountered to understand the complexity of the software. It is difficult to define that what it means that a software to be complex. To capture the different aspects of software complexity; software complexity has been studied and lots of measures have been proposed. Different researchers give different definition of software complexity, some are given below:

- IEEE defines the software complexity as “the degree to which a software component or system has an implementation that is difficult to validate and verify” [4].
- Basili [5] [3] defined software complexity as calculating the expenditure of the resources by a system while interacting with a piece of software system to perform a task. The software complexity is defined in terms of execution storage and time required when computer system is act as an interacting system. And the complexity of software is defined by difficulty of performing different tasks like modifying,

debugging, testing or coding of the software program when we consider programmer (human being) as an interacting system.

- Bill Curtis [6] proposed two types of software complexity:
  a) Psychological Complexity: this complexity affects the performance of programmers trying to modify or comprehend a program unit.
  b) Algorithmic Complexity: It characterizes the run time performance of an algorithm and it is also known as computational complexity.

Thus in literature there is no standard definition exists for software complexity of a system. As different researchers have different views on software complexity that's why it is a multidimensional attribute of software system [5]. Quantifiable measure which is used to measure the characteristics of software development process or system is known as metrics. To check the quality of software which measure that whether the system satisfies the requirement or not, we require a software complexity metrics.

1.4 Need of Study

The important objective during the development of software is controlling and measuring the software complexity. The complexity of the component based system is dependent on the complexity of the already existing components which are used in the development of the software. Coupling is defined as the strength of association which is developed by building a connection between components. The aspects like testability, portability, ease of modification and reusability are affected by the complexity of the components. Thus by using the software complexity metrics we are able to calculate the complexity of the component before using that component in the system. There are several object- oriented coupling metrics aim to measure the relationship between different classes and their methods in a component. But the already existing object- oriented coupling metrics like Coupling between Object (CBO), Response for a Class (RFC) and Depth in Inheritance Tree (DIT) does not consider the relationships while measuring the complexity of the component. It provides the same results for the different components with the same class model (i.e. having equal number of classes) with different connectivity patterns. In this thesis we proposed a coupling metrics which overcome the problem of already existing object oriented coupling metrics. The proposed metrics consider the connectivity patterns

(i.e. UML relationships) between the classes with the help of which we are able to get the unique results for the components with same class model and different connectivity pattern.

1.5 Outline of the Thesis

This thesis has divided into seven chapters whose outlines are given below:

Chapter one describes the motivation for the thesis. It provides the overview of the thesis, brief introduction of the component based software engineering and description of software complexity metrics. It highlights the need of suitable metrics for measuring the component complexity.

Chapter two is a study of CBSE, which provides an explanation of the Component Based Software Engineering approach, its purposes, its terms and concepts, the process models used for the component based software development, the UML relationships and some guidelines for component developers.

Chapter three is a survey on component complexity metrics and interface ability issues, which describes the need of appropriate complexity metrics for components. In this chapter various component complexity metrics and coupling metrics have been discussed with their limitation.

Chapter four provides the problem definition which is to be solved. It also describes the research gap, objectives and the scope of the study.

Chapter five describes the proposed component complexity metrics used for measuring component coupling complexity. This chapter provides the explanation of the proposed metric, its theoretical evaluation, empirical evaluation of the proposed metric.

Chapter six describes the results and analysis of the proposed coupling metric.

Chapter seven discusses the conclusion and future work.

Chapter 2

Component Based Software Engineering

2.1. Introduction

Today software is heart of many organization systems. But software systems are becoming more complex and larger than before. CBSE (Component Based Software Engineering) is the process which uses reusable component to focuses on construction and design of computer based system. This shifts the principle from programming software to composing software system. This embodies the philosophy of "buy, don't build". The CBSE is the process of building the software system for already existing components. Thus it increases the quality and productivity and also reduces the development cost and time [1].CBSE is similar to the concept of Object Oriented Programming (OOP) [7]. In OOPS the different objects are connected together into the program and act as the reusable entity. In the libraries of reusable code different objects are stored. With the help of glue code the components are integrated into the system. A component is an executable program or a function that is not taken as object. But for development of components orientation technology is used. The objects are specified and detailed whereas the components are considered as the standalone service provider. In CBSE the selection, verification and testing phase of software life cycle requires more efforts whereas the component evaluation and selection are the important phase of the life cycle. Due to extensive use of components CBSE is different from the traditional approach like waterfall approach etc.

2.1.1 Component

The components are used via interfaces, it offers the predefined events and able to communicate with other components. A component is used for specific purpose and a self contained unit for the component users [8]. Components are reuse via internet and reused in other environment because the components are distributed as object code not within the organization where it is developed but also outside the organization. The component is developed for reuse in such a way so that it can be deployed or used in other

application with little or no modification. There are different definition given by different researchers, some are given below.

“Components are software elements that conforms a component model and can be independently composed and deploy without modifications according to a composition standards.” [8]

By Councill and Heinmann

“Software component is defined as a unit of composition with contractually specified interfaces and explicit context dependencies, which can be deployed independently and is subject to composition by third party.” [8]

By Clement Szyperski

The components are brought from the components vendors so called Commercial-Off-The-Shelf (COTS) [9] but the components can be developed in house also. In development of new system the use of COTS are increasing. Getting the product to the market fast and reducing the development time the using of COTS components is the best way. To integrate the components of COTS the developer should have the good knowledge of that component. Once the integration of component is done then the conflict of components can be recover through component adaptation. It becomes possible to integrate the components into the system when these conflicts are repaired.

2.1.2 Reusability

“Reusability is the process of creating software systems from existing software components rather than building them from scratch [10].” Many different products for reuse created during the software life cycle ranges from ideas and algorithms. Source code is commonly reused, recently design and source code become popular with object oriented class libraries, design patterns and frameworks. Reusability is a segment of source code which can be used again and again with slightly new functionality or no modification. By using the reusable component implementation time can be reduced. Reuse of component will provides a faster development of application with high quality and reduced cost. There are many cases where reuse leads to failure of software component for example when a component is developed it is tested w.r.t. specific

application but it may cause to failure in the new application. These problems may occur because of lack of tools and methodology, lack of documentation, lack of reuse experience etc. New models for software are emerging with the increase of reuse in software system. The main focus was to provide a way to promote reuse with greater degree of success in organizations.

2.1.3 Factors Affecting Reusability

Different factor that affects the reusability of the black box component are:

i. Parameter Incompatibility

The exchange of data occurs between the components when the components are integrated but sometimes it causes problems to exchange the data between components. The components are from different vendors and the source code is not available in component based software system. So in black box components the functionality of the component is not predictable. With the help of customization interfaces it is possible to modify the black box component which results in reducing the complexity. When the return value of one component is passed as the argument of another component to perform different function but there data types are different then this is known as parameter incompatibility problem. It causes an error when the affected component interacts with the other component functionality, which results in system hanging or wrong output. This error can be reduced by using low coupled component or independent component

ii. Less Ease of Modification and Replacement

For maintenance purposes there is a need for replacing the component or do some modifications in the already existing component, sometimes this may cause a problem. The interaction with the replaced component may arise the requirement of modification in the component. The components are less reusable if they are less modifiable or replaceable. By using the components with less coupling or by using the independent component we can solve the problem of modification and replacement.

iii. Interface Complexity

The software quality attributes like testability, reusability are affected by the complexity, so complexity is the important objective of software development. Complexity of the software should be minimized and controlled. To make a component reusable interface

complexity should be taken into consideration. The component should have the low coupling i.e. components have low outgoing and incoming interactions with the components. Maintenance and integration efforts are minimized by reducing the interface complexity. The components are easily integrated with other component if they have less complex interface, to provide the functionality and made the component more reusable. The interface complexity has been reduced by generating the components with less number of outgoing and incoming interactions or by generating the independent component. With the help of this the reusability of the component is increased.

iv. Test Cases

As in the case of black box reuse the source code is not available so it is difficult to trust the functionality of the reusable component. So to generate test cases and understand the functionality of the component in that case is difficult. Reusability of the component is affected by this. So the appropriate test cases are developed and proper testing reports are provided to the consumers. The time, effort and cost for generating the test cases for the component with low coupling or independent component can be reduced. Because this will generate less number of test cases as there are low number of combination of inputs and their results. Thus generating test cases are convenient for consumers as well as for the developers.

2.2 CBSD Process Model

For assuring the efficiency of software development using CBSD approach several process model have been developed. Some of them are discussed below:-

2.2.1 Somerville's Process Model

A sequential approach has been provided by Somerville for component based software development [11] [12]. There are six phases in this process model as shown in the figure 2.1.

i) **Outline System Requirement:** the requirements of the users are not developed in detail they are developed in outline. As the number of components which are used are might be limited if we use the specific requirements.

ii) **Identify Candidate Component:** in this phase the complete summarize set of specifications are used which help in identifying as many components as possible for reuse.

iii) **Negotiate Requirements:** in this the requirements are modified and refined in order to act in accordance with components.

iv) **Architectural Design:** the architectural designs are developed in this phase.

v) The step 2 and step 3 may be repeated after system architecture design phase. For incorporating compatible components the requirements can be negotiated.

vi) **Integration:** finally to get the software system the selected components are integrated.

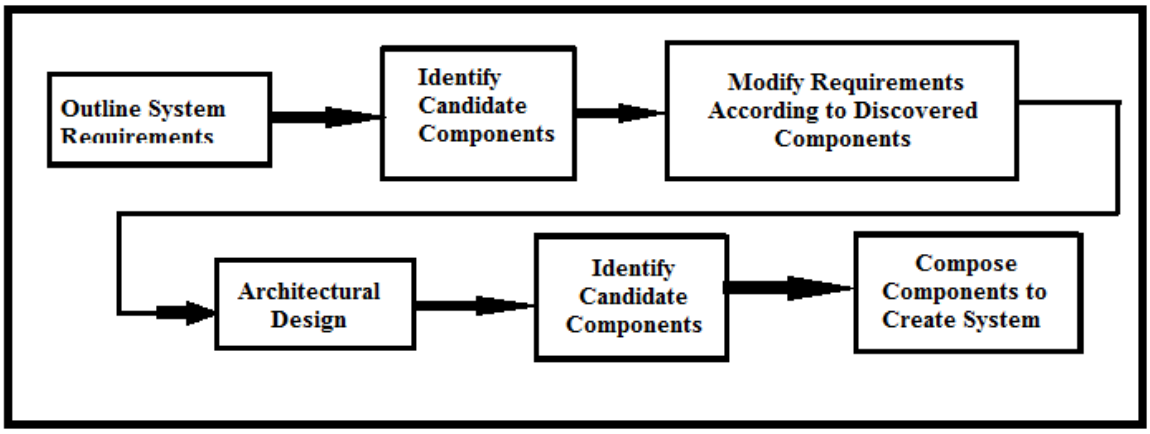

Figure 2.1: The CBSD Process Model Proposed by Somerville [12]

2.2.2 The V- Process model

V model is an adaptation of building a software system from already existing components using traditional waterfall model [11][13][14]. A V model defines a sequential process which consists of following phases as shown in figure 2.2.

i) **Requirement Analysis and Specification:** In this phase the main concerned is with specifying and analyzing requirements keeping in mind the available components present in the pool. For making the use of already existing components the requirements are negotiated, if possible.

ii) **System and Software Design:** In this phase the component pool is available. The components which selected during the first phase may be rejected if the already

available components do not fit in the design of the software system. From the component pool the alternatives may be selected.

iii) **Implementation and Unit Testing:** Building a component from the connection of component interfaces or direct integration is an ideal case for the development of application. But in practice, in order to connect the component interface mismatches some component wrappers or glue code need to be written. Some new functionality need to be added by writing code for filling the gaps in component capabilities and system requirements. Then these components are tested separately.

iv) **System Integration:** This phase is concerned with integration of standard components which build an application components and component framework.

v) **System Verification and Validation:** Standard verification and testing techniques are used here.

vi) **Operation Support and Maintenance:** New components may have been added or already existing components may have been modified/ replaced in the system in order to support the change in the requirements. The maintenance process consists of modifying or deploying of new components, replacing the troubling components or change in glue code which are used for integration of the components in the system.

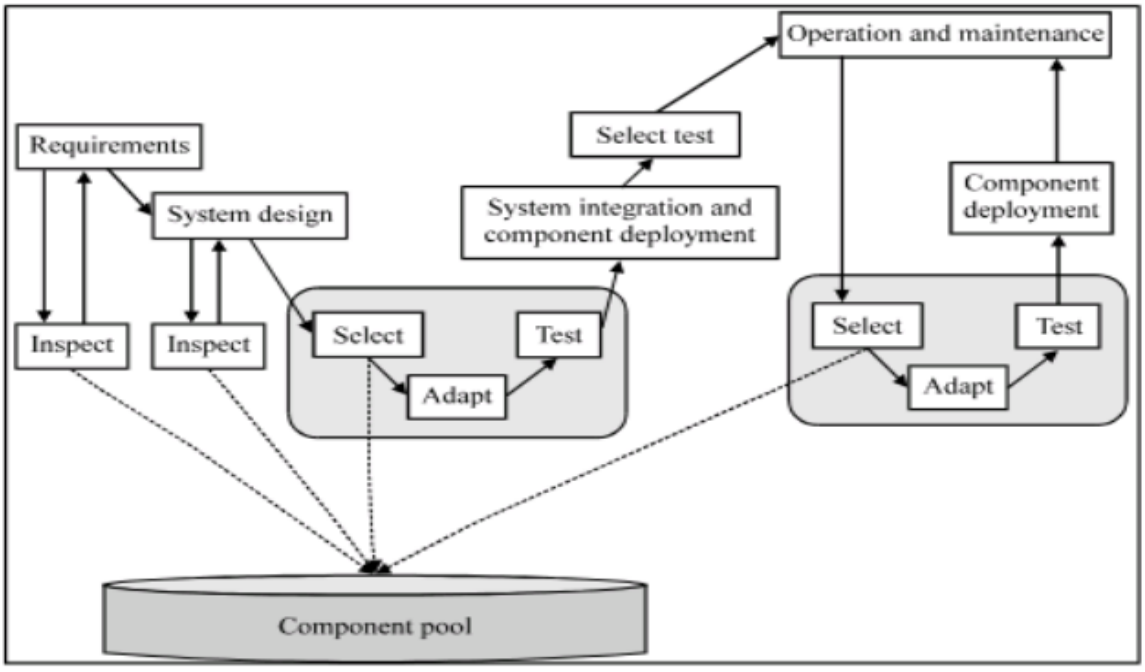

Figure 2.2: The V Process Model [14]

2.2.3 The Y- Process model

The basic activities for the development of the software systems like design, analysis, implementation, deployment, testing and maintenance are contained by the Y process model [11] [12]. To support the component based software development some additional activities are also introduced as shown in figure 2.3.

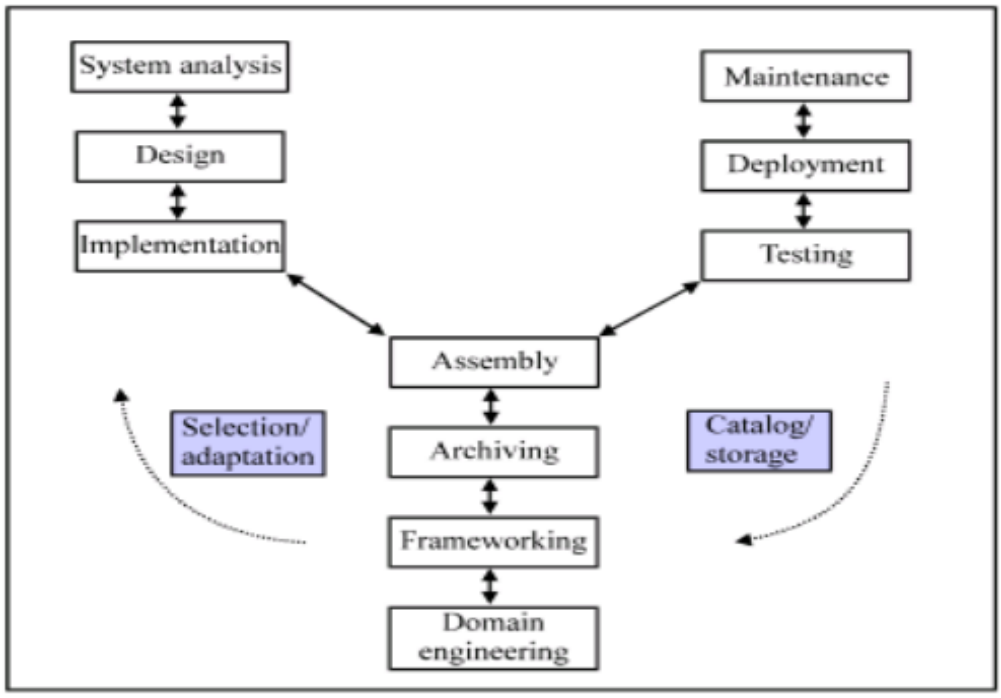

Figure 2.3: The Y Process Model [11]

The additional activities which are introduced to support the CBSD are described below:

i) **Domain Engineering:** this activity is concerned with the identification of the components which are potentially reusable for a particular domain.

ii) **Frame Working:** to produce software in a particular application domain a template or a skeleton is used which is known as framework. For a particular domain the framework captures the semantic relationship between the components. The objective of this phase is to reuse the already existing components and further classify them to form a new framework.

iii) **Assembling:** this phase is concerned for the development of the software application by composing the already existing reusable components.

iv) **Archiving:** this phase is concerned with archiving the components which are developed for a particular domain for future use in associated applications.

Archiving involves different activities such as component storage and cataloguing.

2.3 Coupling

The high quality design of the software system obeys the principle of low level coupling between modules, among many different principles. Coupling is defined as the strength of association which is developed by building a connection between modules. Therefore if there is strong coupling between the modules then there is more difficulty in changing, understanding and maintaining the modules and thus it results the software system to be more complex. The separation of the software system into subsystem is done for the software reuse while for the development of the software system the integration of the subsystems is done. It seems hard to achieve the software reuse and efficient software development on the same time. Higher coupling between the packages result into the lower reusability of the packages because it is difficult to separate the highly coupled packages. It is difficult to maintain the system if it is highly coupled (i.e. make the system complex).

2.3.1 Types of Coupling

To measure the interdependence among different modules Myers, Glenford J [15] defined six different coupling:

1. Data Coupling: if the data is passed through array or scalar parameters between two modules then they are data coupled.

2. Stamp Coupling: If data is passed through parameter i.e. record among two modules are known as stamp coupled. As compare to data coupling the stamp coupling are worse because all modules are affected if the records are changed.

3. Control Coupling: if one module passes the flag value to the other module which is used to control the internal logic.

4. External Coupling: if two modules communicate with each other through external medium then they are known as external coupling.

5. Common Coupling: if two modules refer to the same global data then the modules are common coupled.

6. Content Coupling: if the two modules change and access the internal procedural and data state then they are known as content coupled.

No coupling is known as the "zeroth level coupling" by Offutt et al. [16]. It is considered that if the two modules are coupled using more than one coupling level then the modules is coupled at the highest level.

The notion of tramp coupling is introduced by Page-Jones [17], which is the flow of data through many modules. The flow is from the module to module i.e. where the data are defined and where they are used. It is differ from the other coupling level as instead of just two modules it measures the coupling among different modules.

2.3.2 Principles of Coupling

There are different principles [18] of coupling:

1) The dependency structure of a component which is released must be directed acyclic graph and should not contain any cycle. This principle is known as Acyclic Dependencies Principles.
2) In a design of the components dependency should be in the direction of stability. One component should only be dependent upon another component which is more stable then the first one. This principle is also known as Stable Dependencies Principle.
3) Third principle is the Stable Abstraction Principle, in this principle abstraction of a package and the stability should be proportional to each other. Those packages which are maximum stable should be maximum abstract. The packages are concrete if they are instable.

2.4 UML Relationships

In object oriented design very few classes are stand alone. Relatively large number of classes combines with other classes. The relations are also modeled while modeling object oriented design based on how classes are collaborated to each other. In object oriented modeling according to Booch et al. [19] there are three types of relationships among classes. These relationships are named as association, dependency and inheritance relationships. The structural relationships among the objects are shown by association relationship. The dependency relationship represents the using relationship between the

classes. And the inheritance relationship connects the base class (i.e. generalized class) to their derived class (i.e. specialized class).

i) Association Relationship

It shows the structural relationship between the classes. By playing some kind of roles the classes are collaborated to each other in the association relationship. It is also have a kind of has-a relationship. The association relationship is further divided into two types that are aggregation relationship and composition relationship.

ii) Dependency Relationship

It is a using relationship. When one class uses the functionality of other class, it comes between the classes in UML. In Object Oriented implementation, one class can use the functionality of other classes in different manners. First one is as parameter dependency; in this one class uses the object of other class as a parameter in its method. By looking at class declaration it can be located. Second is Return parameter dependency; in this the object of the class is used as the return parameter of the other class. By going through the method implementation we can identify this type of dependency. Third is the Local Variable dependency which is defined as the object of one class is used in the method of other class as its local variable. It can also be identified by going through the method implementation.

iii) Inheritance Relationship

Generalization is known as is-kind of relationship. Generalization for most of the time is used between the interfaces and the classes to show the relationship of inheritance. This type of relationship exists between the base classes' i.e. general kind of classes and their derived classes' i.e. specific kind of classes. The specific classes also referred to as the child classes or sub classes and the generalized class is known as the parent class of the child class. The significance of generalization or inheritance relationship has been reported by Dirk. He emphasized the object oriented metrics like coupling and cohesion should be studied by considering the effects produced by the generalization or inheritance. This relationship can also be created between the packages.

2.5 Guidelines for Component Developers

To generate the more reusable component, the developer should consider the following points [20]:

i) The components which are develop for the reuse should be generated as a component with less dependency on another component to increase maintainability and reduce the integration efforts.

ii) Generalization of components should be done as specific problem takes more efforts to solve than that of general problem.

iii) The source code should be provided so that it is easy to understand the semantics of the components by the application developers.

iv) As the components are used in different environment so the appropriate test cases should be provided to the user to check that the component is suitable for that environment.

v) Documentation is very important in the reuse; the new user does not have the accurate knowledge of the component in order to use the component with the new requirements. So unambiguous and appropriate documentation having all features should be provided by the developers. Document contains all the properties like resource consumption, robustness, functionality, performance etc.

vi) The components should be certified for its quality, the reuse of already exist component by the user require some faith that the component they are using is reliable and suitable as they are documented.

Chapter 3

Literature Survey

3.1 Introduction

During software development paradigm controlling and measuring the software complexity is an important aspect. The complexity of software affects different attributes like maintainability and testability etc [5]. Researchers proposed different complexity metrics to measure the coupling between the classes and components. CBSD is becoming the trend for developing the software system. Component Based Software Development is based on developing the software system by integrating the prefabricated components from the repository. The quality of complete system is depending upon the complexity of the integrated components. Metrics play an important role in quality assurance specially in deciding whether they should be used or not and in acquisition of components. It should provide the information for deciding whether the reuse of component is sensible or not. In this chapter the various components and coupling complexity metrics have been discussed with their limitations and their proposed solution.

3.2 Study of Various Existing Complexity Metrics

As the complexity of the component affects the different attributes of the software like maintainability, testability etc. so it is important to measure and control the complexity of the software development paradigm. Researchers proposed different metrics but most of them are based on the source code of the software. CBSD is the approach of developing the software by integrating the already existing components. The quality of the new developed system is based on the complexity of the composition of already existing components. So in CBSD the evaluation of complexity is the critical activity of the component. So the suitable metrics are used for the evaluation of the components. In assuring the quality especially in acquisition of components metrics play an important role. It decides whether to use a particular component or not. To decide whether the reuse is sensible or not metric should play an important role. Different metrics are discussed for measuring the component complexity.

3.2.1 Traditional Metrics

To measure the complexity of components in 1970's the traditional software complexity is used. Developers consider Lines of Code, McCabe's Cyclomatic complexity, Halstead's complexity, and Henry's & Kafura's fan-in and fan-out metrics as traditional metrics [21]. Brief description of traditional metrics is given below:

i) McCabe Cyclomatic Complexity Metrics

It is based on program graph and defined as

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

Where:

e represents number of edges in the graph,

n represents number of nodes in the graph and

p represents number of connected nodes in the graph.

ii) Halstead's Complexity Metrics

It is used to estimate the programming efforts. By evaluating the number of operands and operators it measures the complexity of the component. The countable properties are:

n1 & n2 are numbers of distinct operator and operands respectively and N1 & N2 are usage of all operators and operands in the component implementation. The length of implementation of the program is defined as:

$$N = N1 + N2$$

The volume of the program is defined as

$$V = N \log_2 (n)$$

The effort is defined as:

$$E = D * V$$

The difficulty D of the implementation of the program is defined as:

$$D = (n1 * N2) / (2 * n2)$$

iii) Lines of Code (LOC) Complexity Metrics

Lines of Code is used to estimate the size, it is based on the size of methods. This metrics measure the statement, comments or physical lines.

iv) Henry's and Kafura's Complexity Metrics

Henry's and Kafura's proposed a complexity metrics on the basis of the flow of information.

$$\text{Complexity} = (\text{Proc. Length}) * (\text{fan in} * \text{fan out})^2$$

Where Length is Line of Code or McCabe Cyclomatic Complexity

Traditional software complexity is not sufficient of measuring component complexity as measuring of component complexity is based on source code which is not available in case of black box components.

3.2.2 Object Oriented Metrics

Different metrics for object oriented systems [22] are discussed below:

i) Weighted Methods per Class (WMC)

Weighted Methods per Class is used to measure the complexity by counting the number of methods in a class. It is implemented within the class and measured by cyclomatic complexity. It is difficult to measure the complexity of component by counting all the method due to inheritance. How much effort and time is required to maintain and develop the class is dependent on the complexity of the class and the number of methods. As all the methods are inherited by the children in the derived class, so it have greater impact on the children as it have larger number of methods in a class. As the value of complexity metrics increases the understandability decreases and testing efforts increases.

ii) **Response for a Class (RFC)**

Response for a Class is used to calculate the component complexity by counting the sets of method invoked by the object of the class. Response for a Class includes methods which are assessable within the class hierarchy. The calculation is done on the basis of the amount of communication with other class or by the number of methods in the class. The complexity of the class is greater if the methods invoked in the class through messages are greater.

iii) Lack of Cohesion (LCOM)

Lack of Cohesion (LCOM) measures the complexity of the class by measuring the dissimilarity of methods in a class. It measures the dissimilarity by the different attributes or instance in a class. If a class is highly cohesive than that class should be stand alone.

We get high cohesion and high simplicity by increasing the cohesion. Single class with low cohesion is subdivided into different classes with more cohesion.

iv) Coupling between Object Classes (CBO)

Coupling between Object Classes (CBO) measures the complexity of the class by counting the number of class coupled with other class. It calculates complexity by counting the number of non inheritance class hierarchies which depends on another class. If there is excessive coupling between the classes then it prevents reuse. To reuse the class the class should be less dependent on other class. If the classes are highly coupled then it is difficult to do modifications in the class and therefore maintenance of the class is difficult. If the classes are strongly coupled then it becomes difficult to understand the class. To reduce the complexity between the classes it is necessary to reduce the coupling, which promotes encapsulation and improves modularity.

v) Depth of Inheritance (DIT)

The maximum number of steps from root node of the class to the end is the depth of the class with the hierarchy of inheritance. The depth of class is measured by the number of ancestors in the classes. If the depth of the class is greater in the hierarchy i.e. contains maximum number of methods make it more complex to predict its behavior. Design complexity becomes greater if the depth of the tree is greater. More the classes and methods greater the potential for reuse of the class and methods. The number of methods and class inherited are the support metrics for the depth of inheritance.

vi) **Number of Children (NOC)**

To measure the complexity of the class the number of children in the class is counted. In a hierarchy the number of subclasses in a class is the number of children. On the system or design the number of children is the indicator of potential influence. Greater number of children causes misuse of sub classing and also the improper abstraction of the parent class. Also if we have greater number of children in the class then we have greater reuse, inheritance is a form of reuse. It requires more testing of the methods of the class if we have large number of children and it also increase testing time.

3.2.3 Component Complexity Metrics

For determining the component complexity many researchers have proposed various metrics, some of them are discussed below:

a) Tullio. Vernazza et.al extended the Chidamber & Kemerer Metrics [23]. The definition of new metrics which were proposed corresponding to each CK metric have been given below:

i) Extensions for DIT

To identify the classes the high value of DIT is used that are hard to maintain. The value of DIT indicates the effort in maintaining a group of classes. But the highest value of DIT (MAXDIT) and the mean of DIT of unrelated trees (MUT) are considered by the extended metric. MAXDIT is defined as:

$$MAXDIT = \max_{Ci€k} \{DIT(Ci)\}$$

Where

k consists of m classes in a component and DIT (Ci) value of i^{th} class which belongs to component k. The mean values of MAXDIT for each class hierarchies are the mean of DIT of unrelated trees. Assume that k is partitioned in r unrelated components U_i …. U_r with:

$$\coprod_{i=1}^{r} Ui = k$$

Ui and Uj never share any super class when i and j are different. Thus MUT is defined as below:

$$MUT = \sum_{i=1}^{r} MAXDIT\ (Ui)$$

ii) External Coupling between Objects (EXTCBO)

It is an extension of CBO. The level of coupling for a component is measured by EXTCBO. The number of external classes is given by it which is coupled to the component. It is defined as:

$$EXTCBO = \sum_{i=1}^{m} ei$$

Where

m is the number of classes in a component and e_i is the external classes coupled to the class Ci.

iii) Response Set for a Component (RFCOM)

It is an extension of RFC (i.e. Response for a Class). It gives the number of methods called by the member classes and the number of methods in that classes. High value

of RFCOM gives high complexity of the component. It is the sum of values of RFC for all classes and is defined as:

$$\text{RFCOM} = \sum_{i=1}^{m} RFC\ (Ci)$$

where

m number of classes in a component and RFC (Ci) value of Response for a Class Ci.

b) Different component metrics were proposed by Cho et al. to measure the complexity, reusability, customizability. These metrics can be used to measure the characteristics of the components [22]. Cyclomatic complexity combined with new component complexity metrics to calculate the complexity of the component. CBSE has been largely focused on component modeling, methodology, architecture and platform. As the demand of component increasing in market, it becomes important to plan metrics to determine the quality of various components. The new Component Complexity Metric is defining into four types of complexity metrics: Component Plain Complexity (CPC), Component Static Complexity (CSC), Component Dynamic Complexity (CDC), and Component Cyclomatic Complexity (CCC). The line of code are good for measuring the reusability but function points are not suitable for measuring both the complexity and the reusability, as function points are not suitable for component based software development.

c) Washizaki *et al.*'s proposed a software metrics for measuring the reusability of Black-Box Components [24]. Measuring the reusability of component is necessary in component-based software development in order to reuse the components effectively but due to unavailability of source code it is difficult. The set of metrics were defined which were compliant to Java Beans Component model to measure the reusability property of black box components. Different metrics were defined to measure the attributes of the component which contributes into the reusability of the component, they are: existence of Meta information, customizability, observability and external dependency. Different proposed metrics set includes: Existence of Meta-Information (EMI), Rate of Component Observability (RCO), Rate of component Customizability (RCC), Self-Completeness of Component's Return Value (SCCr), Self-Completeness of Component's Parameter (SCCp) and Component Overall Reusability (COR). RCO and RCC have very strong positive relation with each other. Similarly SCCr and

SCCp were positively correlated with each other. This concludes that the metrics proposed here was used to measure the reusability of the component. The analysis tool accept only JavaBeans component, it will be extended to accept other components also.

d) Boxall defined different interface metrics are developed which shows that proper measurement of component interfaces can provide relevant information for the reusability of the component [25]. As Component interface metrics increases our understanding of the reusability of the components. As different source of information which are relevant for reuse is not available as that tends to be black box component. There automation provides more accurate and efficiency of the reusability of the component. The different metrics brought a reusability analysis of component which was consistent with expert knowledge. It considers that interface metrics may be able to be used for reusability of components with which the metrics is unfamiliar. These metrics help in quantifying the reusability of the component.

3.2 Interface ability Issues

Stevens et al. [26] defined coupling “the measure of the strength of association established by a connection from one module to another." The classes which are highly coupled are not desired as they are considered as bad design. The complexity of the classes increases as the coupling degree increases which results the module to be dependent on the external classes. Coupling measure is a very interesting and rich part of research work, which results in many approaches like dynamic coupling measures, coupling metrics for knowledge based systems, structural coupling metrics, information entropy coupling measure approach, logical and evolutionary coupling, etc. There are different structural coupling metrics which received significant attention in the literature. CBO (Coupling between Objects) and CBO_1 were defined by Chidamber and Kemerer [27] as method of one object using instance variables or methods of another object. Due to inheritance CBO metric excluded coupling. Several metrics were empirically validated and were useful for predicting fault-prone classes. Other structural metrics are RFC (Response for Class) and RFC∞, COF (Coupling Factor), DAC (Data Abstraction Coupling), Ce (Efferent Coupling), ICP(Information-flow-based Coupling). Coupling

measures listed above are based on attribute references and method invocations. Briand et al. [28] defined the suite of measures like IFCAIC, FCAEC, OCAIC, ACAIC; etc captures different types of interactions like class- method, method- method, and class-attribute interactions.

a) Hitz and Montazeri [29] proposed different types of coupling, they were class and object level coupling. These were determined by state of object's implementation like class interface at a given time in software development cycle and also by state of object i.e. value of its attributes at runtime. State dependencies between class's results in Class Level Coupling (CLC) in the system while state dependencies between objects at runtime of a system results in the Object Level Coupling (OLC). Class Level Coupling Also determined by static analysis of source code. OLC is important when considering change dependency and maintenance. CLC depends on object structure which is determined by input data; therefore it is input data and source code at runtime. CLC is relevant for debugging and testing.
b) Li and Henry [30] discussed class coupling in terms of Message Passing Coupling (MPC) and Data Abstraction Coupling (DAC). He identifies different metrics to predict the maintainability of the system design.
    i) MPC is measured as the number of method invocations in a class.
    ii) DAC is defined as the number of attributes in a class that have another class as their type.
c) R.Harrison et al [31] discussed two coupling metrics: the Coupling between Object and the NAS metric. NAS metric developed using the GQM (Goal Question Metric) approach. There is a strong relationship between the NAS and CBO metrics. It implies that only one of them is needed to access the level of coupling at design time. NAS metric is easily available at design time and easy to interpret to predict early coupling. There is no relationship found between the understandability of the systems and the class coupling.
d) To measure the quality of an OO design R.Martin proposed two coupling metrics [32] that are Afferent Coupling (Ca) and Efferent Coupling (Ce). It calculates in terms of interdependency between the subsystems.

i) Afferent Couplings (Ca): The other packages that depend upon different classes within the package are an indicator of the responsibility of the package.

ii) Efferent Couplings (Ce): The number of different packages that the number of different classes in the package depends upon is an indicator of the independence of the packages.

He does not specify the dependencies between the classes exactly. To predict the modification effort, new coupling measures were developed in the context of amount of modification of code need to extend the functionality of components and in the context of time required. There are already metrics available for the modification efforts they are CBO, RFC, DAC, MPC, and COF. Using these static metrics, they show significant relation with reuse effort. But also there are some limitations that metrics fell short of their performance and perfect ranking which was not consistent for different components.

e) Haun Li [33] proposed a new metric known as global coupling metric. There exist different metrics but they have some limitations. There are 2 aspects in which the proposed metric is differing from existing metrics:

i) It reflects the indirect coupling between the classes.

ii) It takes into consideration the strength that one class is dependent on another class.

In this global metrics it reveals the indirect and direct coupling interaction. A representative object oriented software system has been compared and analyzed with an existent CBO metric. The overall result shows that global metrics reveals interaction relation more precisely and deeply than CBO metric. It require to keep the strength of global coupling metrics small, as small value of it shows that the measured class is apt to be debugged and changed. Even there is large acceptance of object oriented programming, but some programmers does not have a good grip on the intimate mechanism and the design principles of object orientation which results in a poor designed system. To measure the design performance of the system coupling is the most vibrant quality attribute.

Chapter 4

Problem Statement

4.1 Research Gap

In Object Oriented Metrics, there are certain metrics those define different aspects of coupling. These are Coupling between Objects (CBO), Response for a Class (RFC) and Depth Inheritance Tree (DIT). But ambiguity arises where we take into account the same models i.e. the component with same number of classes. These metrics give same value or result when applied on same number of classes in a components. That is these metrics only consider number of classes in collaboration but do not specify type of connectivity or coupling among them. So, it underestimates the significance of interconnection i.e. number of relationships and type of relationships.

4.2 Problem Definition

Coupling is defined as the strength of association which is developed by building a connection between classes. Several object- oriented coupling metrics aim to measure the relationship between different classes and their methods. There are different ways in which classes are connected. These class connectivity patterns give the structural view how the classes are linked and connected. Object oriented metrics like Coupling between Objects (CBO), Response for a Class (RFC) and Depth of Inheritance Tree (DIT) are some of the metrics that are applicable to classes. These metrics measure various aspects considering relationship between classes, methods and inheritance etc. These metrics obtain their value solely depending upon number of classes under consideration. There can exist two models with same number of classes in them which mean the two models are same. But instead of having equal number of classes in them, these models can be identified as distinct on the basis of their linkages as one model has more connectivity or coupling among classes than the other model. It means same models with different number of connections or link patterns obtain the same value of CBO, RFC and DIT but the connectivity pattern clearly indicate that the degree of coupling are different. Object oriented coupling metrics like CBO, RFC and DIT do not take into consideration the relationship among classes and ignores underlying coupling complexity. This anomaly

shows the inadequacy of above metrics in differentiating the same class models w.r.t the coupling or connectivity patterns. The underlying problem is referred to as Inability to Differentiate Anomaly (IDA).

4.3 Objectives

i. The detailed study of Component Based Software Engineering (CBSE), various terms and concepts of CBSE, Software Complexity Metrics and UML relationships.
ii. Identification of problematic software metrics w.r.t their connectivity patterns.
iii. Analysis of different connectivity patterns among classes in components.
iv. Proposal of coupling complexity metrics for components.
v. Empirical evaluation and validation of the proposed metrics.

4.4 Scope

Coupling based on Strength Specification (CSSM) metric find its scope in industry. It is useful in visualizing the software for better designing, coding and reusing. It also gave a quantifiable measure that can be used to understand object oriented software in terms of numbers and types of interconnection and interrelationships. Using this metric, software industry those are using Object Oriented system can deliver qualified designs and code.

Chapter 5

The Proposed Metric:

Coupling based on Strength Specification (CSSM)

5.1 Introduction

A major factor that affects the component complexity is coupling. It refers to the interdependency between the components in a software system. The coupling has been correlated with important quality attributes like robustness, maintainability and reusability [4]. To understand the system, coupling metrics proves to be an aid in understanding system by tester, developer and maintainer. To evaluate the influence of changes in components, to identify the critical components and to support the future evolution of the component based software system while adding, removing or modifying some components. Component complexity is an essential metrics to be considered at the specification phase the complex and large systems are evaluated to avoid poor interaction in the components of the software system. Also the faults and failure of a component which leads to a failure of complete system can be evaluated using coupling metrics. In this chapter a coupling complexity metric based on strength specification has been proposed. The foundation of the coupling metrics is basically built upon the proposed metric which is capable of evaluating the system, based on same class models. It effectively gives unique results in measuring coupling strength of two or more components having same class model.

5.2 Problem in Already Existing Object Oriented Metrics.

The components under consideration are the components which are collection of classes and their relationships. The important object oriented metric suites are given by Chidamber and Kemerer (CK) [34], Briand, Hitz and Montazeri [29], MOOD Metric.

A) Relationships between Classes

Here in Chidamber and Kemerer (CK) Metrics the main metrics those are qualified for giving actual picture of classes and their relationship are; Coupling between Objects (CBO), Depth Inheritance Tree (DIT) and Response for a Class (RFC). These metrics

directly or indirectly define coupling or give the degree of strength with which the classes or components are glued. But, these are unable to define the strength of coupling on the basis of:

i. Number of Relationships or Connections
ii. Type of Relationships or Connections

In object oriented systems, there exist certain relationships between classes like aggregation, composition, dependency, generalization, etc. These relationships specify particular kind of design constraint on the collection of classes or components. These design constraint on relationships can be taken into consideration to define the strength with which the classes are glued. And also, the different kind of relationships depicts how strongly and loosely the classes are joined or coupled together.

B) Same Class Models

If two components are having same number of classes then they are defined as same class model. The two different components lying under the same model can be categorized on the basis of coupling strength or glueness. Now in case of object oriented metric the Coupling between Objects (CBO), Depth Inheritance Tree (DIT) and Response for a Class (RFC) are enable to give an accurate picture of the relationship or the coupling between the classes.

C) Inability to Differentiate: Anomaly in Existing Object Oriented Metrics

Inability to Differentiate Anomaly is defined as "the same models with different number of connections or link patterns attain the same value of Object Oriented Metrics like Coupling between Objects (CBO), Depth Inheritance Tree (DIT) and Response for a Class (RFC)."

Table 5.1 defines Coupling Metrics: Coupling between Objects (CBO), Depth Inheritance Tree (DIT) and Response for a Class (RFC). These metrics are considered to showcase an anomaly (i.e. the Inability to Differentiate Anomaly).

Table 5.1: Definition of Object Oriented Coupling Metrics

Metrics	Description
Coupling Between Objects	Coupling between objects (CBO) is a count of the number of classes that are coupled to a particular class i.e. where the methods of one class call the methods or access the variables of the other. CBO was defined by Chidamber & Kemerer [34].
Response for a Class	Response for a Class (RFC) [35] is defined as the set of methods that can potentially be executed in response to a message received by an object of that class. All the methods in the class and all the methods that are called by methods in that class.
Depth of Inheritance Tree	Depth Inheritance Tree (DIT) is defined [36] as the count of the classes that a particular class inherits from. It counts the level of subclasses affecting a class. When the class in the hierarchy is deeper then there are more variable and methods to inherit and make them more complex to inherit.

i. Elaboration of Problem in Coupling between Objects (CBO):

The definition of the CBO [34] for a class is the count of the number of other classes to which a class is coupled. The classes which are directly connected are only counted while calculating the metrics. Even if the reference operates in both directions, each class is counted only once. For three reasons the value of CBO should be as low as possible:

a) The dependency between the classes increases the coupling which makes the code less suitable and modular for reuse.
b) The code becomes more difficult to maintain when there is high level of coupling; since an alteration to code in one area affects the code in another linked area.
c) The more the connection between the classes the more complex the code and more difficult to test.

Figure 5.1 show two components having the same class model. The rectangle box in figure represents the classes, circle represents the methods and line shows the link between the classes. Following are the underlying perspective problematic cases in Coupling between Objects (CBO). Steps followed to identify problem in CBO:

Step 1: **Classification:** Consider two components with same class model (i.e. having same number of classes) as shown in figure 5.1. Here we have same class model but number of interconnection are different in Case I and Case II. Case I has connection between C & D and B & E but Case II do not have, yet they are same class model.

Step 2: **Application of CBO Metrics:** Now apply the CBO Metrics on both components. We get the same results for both the components; which shows that both the components are equally coupled.

CBO for Case I	CBO for Case II
2	2

Step 3: **Problem Identification:** But these models can be identified as distinct on the basis of their linkages as one model (i.e. Case I) has more connectivity among classes than the other model (i.e. Case II).

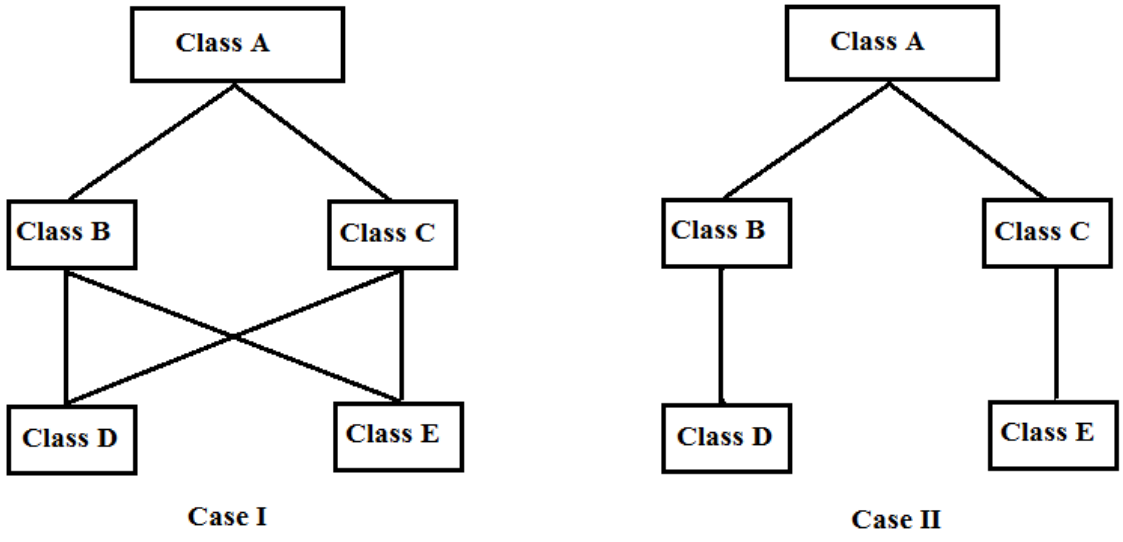

Figure 5.1: Cases for CBO Metrics

Thus Case I is more coupled then Case II. It means that same class models with different number and type of interconnection obtain same value of metrics but the connectivity patterns clearly indicate that the degrees of coupling are different. The underlying problem is termed as the Inability to Differentiate Anomaly for CBO.

ii. Elaboration of Problem in Response for a Class (RFC):

Through the amount of communication with other classes and number of methods the RFC [35] looks at the complexity of a class. It can be represented as:-

$$RS= \{M\} U_{all\ i} \{R_i\}$$

where

$\{R_i\}$ is the set of methods called by method i and $\{M\}$ is the set of methods in the class.

Figure 5.2 show two components with the same class model. The rectangle box represents the classes, circle represents the methods and line shows the link between the classes. Following are the underlying perspective problematic cases in Response for a Class (RFC). Steps followed to identify problem in RFC:

Step 1: **Classification:** Consider two components with same class model (i.e. having same number of classes).

Step 2: **Application of RFC Metrics:** Now calculate the complexity of the two components using the RFC. It gives the same result for both the components.

RFC for Case I	RFC for Case II
4	4

Step 3: **Problem Identification:** But these models can be identified as distinct on the basis of their linkages as one model (i.e. Case I) has connectivity between methods where as in the second component (i.e. Case II) the connectivity is between the classes (i.e. by passing the complete class like composition which show more coupling between classes).

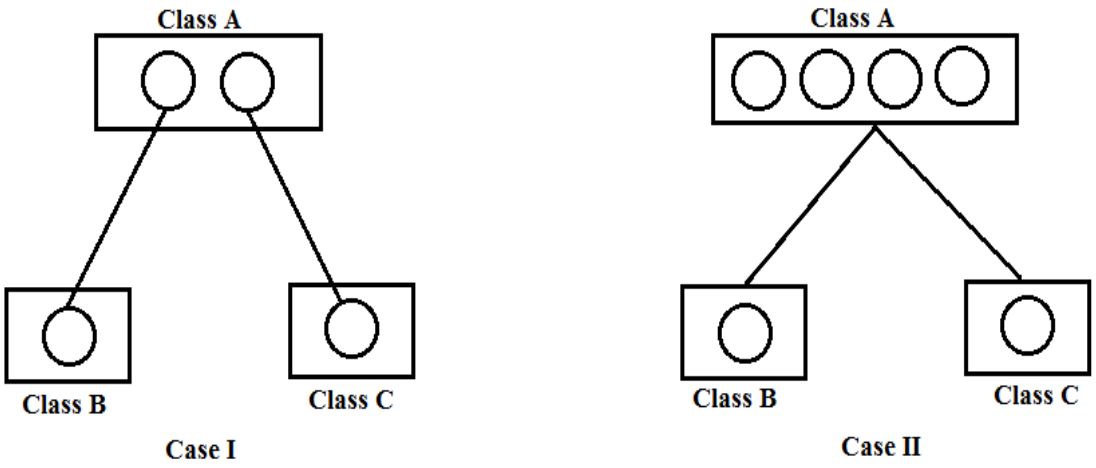

Figure 5.2: Cases for RFC Metrics

Thus Case II is more coupled then Case I. If we obtain the same coupling value for same class model but with different linkages or connectivity patterns, then such cases are referred to as Inability to Differentiate Anomaly for RFC.

iii. Elaboration of Problem in Depth Inheritance Tree (DIT):

DIT is defined [36] as the maximum inheritance from the class to the root class. It counts the level of subclasses affecting a class. When the class in the hierarchy is deeper then there are more variable and methods to inherit and make them more complex to inherit. The tool to manage the complexity of the classes is inheritance. Because of method inheritance deep trees promote reuse. Following are the underlying perspective problematic cases in Depth Inheritance Tree (DIT). Steps followed to identify problem in DIT:

Step 1: **Classification:** Consider two components with same class model (i.e. having same number of classes) as shown in figure 5.3.

Step 2: **Application of DIT Metrics:** Now calculate the complexity of the two components using the DIT. It gives the same result for both the component from Class A.

DIT for Case I	DIT for Case II
2	**2**

Step 3: **Problem Identification:** These models can be identified as distinct on the basis of their connectivity patterns. In the figure 5.3 it is clear that Case I i.e. first component is more coupled then the Case II i.e. second component.

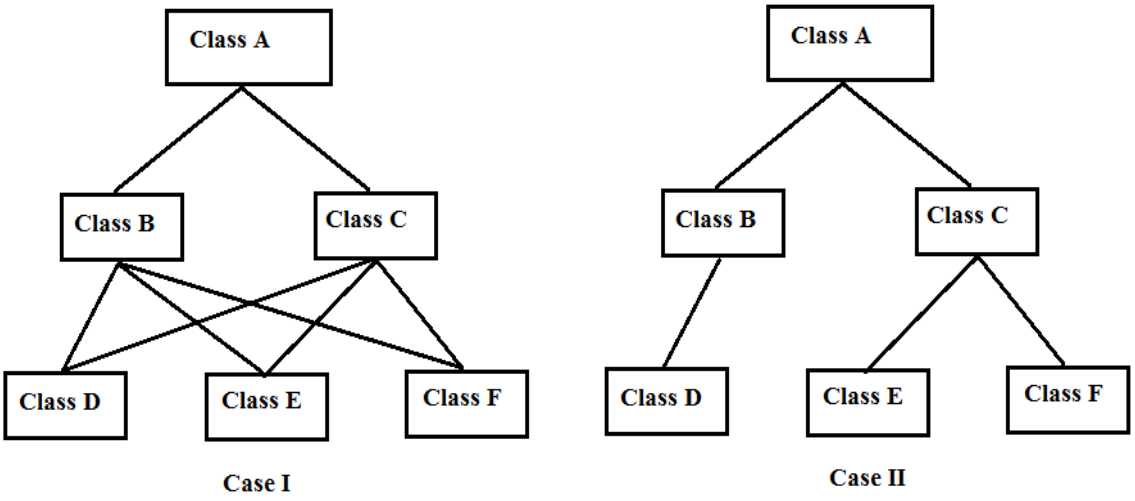

Figure 5.3: Cases for DIT Metrics.

Thus Case I is more coupled then Case II. If we obtain the same coupling value for same class model but different connectivity pattern, such cases are referred to as Inability to Differentiate Anomaly for DIT.

5.3 Evaluating and Ranking UML Relationship according to Degree of Coupling.

Firstly we have ranked all different kind of existing coupling like local variable, parameter type, etc. on the basis of the strength or degree of coupling. Coupling at the granular level in a program code can be identified on the following parameters:

i. local variables existing in the class,
ii. parameter passed in the method,
iii. the type of data return,
iv. the class inheritance,
v. global variable,
vi. composition class defined in another class having life dependency,
vii. aggregation class is defined under another class without any life dependency.

These coupling relationships among classes can easily be shown with the help of UML relationships e.g. generalization, composition, dependency, aggregation, etc. Secondly weights are assigned according to the ranks based on coupling strength.

5.3.1 Evaluation of Ranks w.r.t Strength of Coupling

We have considered different UML connectivity patterns. We define all the patterns in a ranking order i.e. from best case to worst case. It means that the local variable dependency is the best coupling (show low level of coupling) whereas the composition is the worst coupling (show high level of coupling). They are discussed below:

1) Local Variable

Local variable coupling type is defined as the method of one class uses the object of other class as its local variable [37]. By going through the method implementation we can identify this type of dependency.

Table 5.2: Operation on X has Local Variable of Type Y

X:: method Y() {
…..
Y LocalVarTypeY;
…..
…..
};
	Class Y {
	
	.....
	…..
	
	};

Evaluation of Strength of Coupling in Local Variable:

This Local variable coupling is lowest coupling, as here only local variables are accessed from one module to another.

Strength of Coupling in Local Variable	0.05

2) Parameter Type

It also comes under the UML dependency relationship. In this type of coupling a class uses an object of other class as a parameter in one of its member method. By looking at class declaration parameter dependency can be located [37]. In this parameter type coupling we can pass the complete class or member as a parameter to another class.

Table 5.3: Operation on X has Parameter of Type Y

Class X {	Class Y {
…..	
public;	int y
void method_X(Y param); // case m (weight 0.15)	method_y();
void method_X(y); // case n (weight 0.10)	…..
void method();	
…..	};
};	

Evaluation of Strength of Coupling in Parameter Dependency:

In the given figure the parameter are pass between the classes as the complete class (i.e. case m) or as a member (i.e. case n). By simply looking to the class declaration we can locate the parameter dependency. Thus this is less coupled then the return type dependency.

Strength of Coupling in Parameter Dependency (Passes Variable as Parameter)	0.10
Strength of Coupling in Parameter Dependency (Passes Object as Parameter)	0.15

3) Return Type

This type of dependency is defined as the method of one class uses the object of other class as a return parameter [37]. By going through the method implementation we can identify this type of dependency. As shown in figure method of class A uses the variable of class B in its implementation. In this the method of class A i.e. method_A returns the object of class B. In this we have three different cases i.e. case i, j, k which represents the return of object, variable and data structure respectively.

Table 5.4: Return Type Dependency

Class A	class B
{	{

B b;
public:
class B method_A(); // case I (weight 0.50)
int method_A(); // case j (weight 0.35)
int[] method_A(); // case k (weight 0.25)
}
class A :: method_A()
{
b.method_B();
….
return B;
}
	method_B();
	….
	}

Evaluation of Strength of Coupling in Return Type Dependency:

The method of one class uses the object of other class as a return type. We can identify this type of dependency by method implementation; means it is more coupled than the other dependency relationship like parameter type, local variable relationship etc. and it is less coupled then the global variable relationship.

Strength of Coupling in Return Type Dependency (Passes Variable as Return Type)	0.25
Strength of Coupling in Return Type Dependency (Passes Data Structure as Return Type)	0.35
Strength of Coupling in Return Type Dependency (Passes Object as Return Type)	0.50

4) Global Variable

This type of coupling comes under the dependency relationship. It uses a "using relationship" [38]. It comes when one class uses the functionality of other class through

instances. It is also known as non-local, common and shared coupling. It refers to the variable which is defined in one class and is used in other class. In the modern object oriented languages the global coupling is affected by the access specifier.

Evaluation of Strength of Coupling in Global Variable:

The global variables are the variables which are declared outside the block. They have the program scope; means they can be accessed anywhere in the program. When the program ends then the global variables are destroyed. The primary reason to avoid the global variable is that they increase the complexity of the program. We have to examine the every single line in the file to check that for what the global variable is used for. Also the values of global variables can be changed by any function that is called. It is more coupled then other dependency (like parameter type, return type, local variable type, etc) but less coupled then thee generalization relationship.

Strength of Coupling in Global Variable	0.60

5) Generalization

It occurs when one class is a descendant or subclass of another class. The coupling is not made through redefined data members of the base class by its derived class; it is simply made through inherited [37]. Figure shows that class Y and its derived class X. Y class define a data member y and X class inherits y from class Y which is base class. Which shows classes Y and X are inheritance coupled through y.

Table 5.5: Generalization Relationship

```
public class Y
{
…..
…..
}
public class X extends class Y
{
….
}
```

Evaluation of Strength of Coupling in Generalization:

It does not redefine the data that's why it is less coupled then the aggregation relationship. It is more coupled than the dependency relationship coupling as it inherits the complete class or the member functions of the class.

Strength of Coupling in Generalization	0.75

6) Aggregation

An aggregation relationship is a specific type of composition relationship where no ownership between the sub objects and the complex objects are implied. Aggregation relationship [39] shows classifiers as subordinate to another classifier. In this objects are configured or assembled together to create more complex objects.

Evaluation of Strength of Coupling in Aggregation:

In the aggregation relationship when an aggregate is destroyed then the sub objects are not destroyed. This relationship shows the high level of coupling between the classes but shows less coupling then the composition relationship. Because in composition relationship when the composition is destroyed then the sub objects are automatically removed but in case of aggregation when an aggregate is destroyed then the sub objects are not destroyed. The subclass objects are defined outside the scope of class, so when the class is destroyed, the member variables will be destroyed but the sub objects will still exist.

For example:

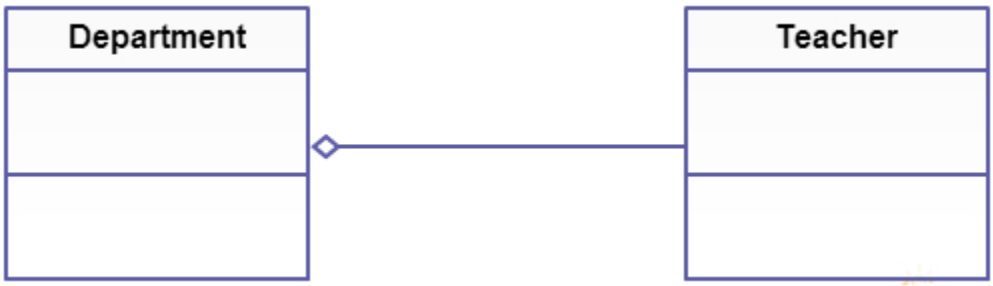

Figure 5.4: Aggregation Relationship

Consider a department of math in a school, which have one or more teachers. The department does not own the teachers; it means that the department is aggregate. So when

we destroy the department then teachers should not be destroyed and should exist independently. Thus aggregation relationship is also referred as weak association.

Table 5.6: Aggregation Relationship between Class X and Class Y

class X	class Y

{
private:
Y.y;
};
	{
	public methody();
	....
	}
	};

The weight assigned to the aggregation relationship is less than the weight assigned to the composition; because the composition is more coupled then the aggregation relationship.

Strength of Coupling in Aggregation	0.85

7) Composition:

Composition association is a form of aggregation relationship. It represents a whole-part relationship, which specifies that the life of the part classifier is completely dependent on the life of whole classifier. The data flow is unidirectional in this relationship. In real life time by using the smaller and simpler objects the complex objects are built. A personal computer built using a motherboard, a CPU, memory, etc. Thus the process of building complex objects from smaller and simpler ones is called object composition. In composition relationship, by using either pointers or normal variables we add our subclasses to the composition and all the process of creation and destruction is handled by the composition class.

Evaluation of Strength of Coupling in Composition:

This relationship represents the whole-part relation; it means that one class is completely dependent on another class. Compositions [40] are complex classes that contain other subclasses as member's variables. The complex object owns all the objects of subclasses which it is composed of. In this relationship if the composition is destroyed then all the other classes which are dependent on this complex class are also get destroyed.

For Example:

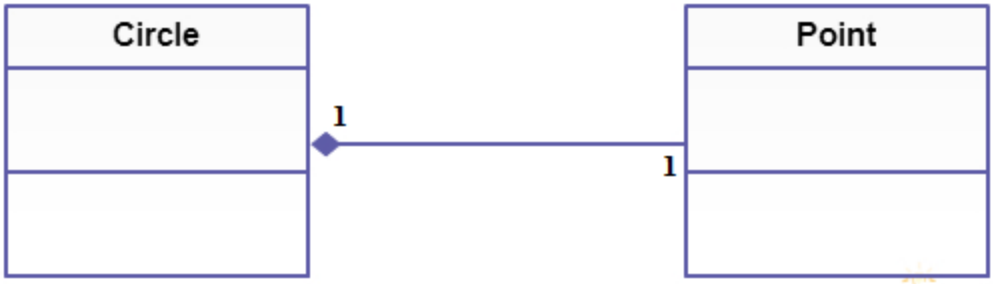

Figure 5.5: Composition Relationship

Consider a class Circle it uses another class named as class Point. The class Circle uses the class Point to denote its origin. Point class is not only the part of the Circle class, but also they both have the same lifetime. It means the Point class is destroyed together with its corresponding class Circle. The composition is also referred as strong association.

Table 5.7: Composition Relationship between Class X and Class Y

class X
{
public:
Y.y1;
};
	class Y
	{
	public methodY();
	{
	……..
	}
	};

Thus this shows that the composition is a very highly coupled relationship.

Strength of Coupling in Composition	0.95

5.3.2 Assignments of Ranks to Coupling Parameter Represented as UML Coupling Relationship:

Coupling parameters are arranged in decreasing order according to their weights and ranks. Strength of coupling is as rank of coupling 1, 2, 3, and so on form high to low level. Strength of coupling can also be referred to as weight assigned to coupling parameters. The table 5.8 shows the weight assigned to the connectivity patterns (i.e. UML Relationship) according to the coupling level from high level coupling to the low level coupling (i.e. from worst to best coupling).

Table 5.8: Ranking Table of Connectivity Patterns

Ranks	Coupling Parameters	Strength of Coupling
RC1	Composition Relationship	0.95
RC2	Aggregation Relationship	0.85
RC3	Generalization Relationship	0.75
RC4	Global Variable Dependency	0.60
RC5	Return Dependency (Return object of other class)	0.50
RC6	Return Dependency (Return Data Structure)	0.35
RC7	Return Dependency (Return variable of other class)	0.25
RC8	Parameter Dependency (Pass Parameter as Object)	0.15
RC9	Parameter Dependency (Pass Parameter as Variable)	0.10
RC10	Local Variable Dependency	0.05

5.4 Coupling Based on Strength Specification Metric (CSSM): The Proposed Metric

There is an acute need of a metric that can measure the strength with which the classes in components are connected. The existing problem in Object Oriented Metrics i.e. CBO, RFC and DIT is considered and a novel metric is proposed i.e. Coupling based on Strength Specification (CSSM). It uses the connectivity patterns to remove the underlying problem which is referred to as Inability to Differentiate Anomaly. CSSM is based on:

a) Existing relationships types in design/ code.
b) Existing number of connectivity pattern in design/ code.
c) Same class models.

Definition:

Coupling based on Strength Specification Metric (CSSM) is defined as the sum of weighted connections or relationships based on the strength of coupling existing among classes.

$$CSSM = AR_c + GR_c + DR_c \quad \text{…. (i)}$$

Where

c represents the classes in the components

AR= number of association relationship between the classes.

GR= number of generalization relationship between the classes.

DR= number of dependency relationship between the classes.

A structural relationship is represented by association that connects two classes. The properties of the classifiers like classes are recorded by associations. To show the design decisions which are made for the classes are can be done by association. We can calculate the Association Relationship (AR) using the equation (ii). It contains both the composition and aggregation relationship.

$$AR = \sum(ni * wa + ni * wc) \quad \text{…. (ii)}$$

Where

n_i = the number of times the particular type of relationship occurs between the classes.

w_a = the weight assigned to the aggregation relationship.

w_c = the weight assigned to the composition relationship

A relationship in which one model element i.e. the child is based on another element i.e. the parent is known as generalization relationship. This relationship can be used in classes, components, etc. We can calculate the Generalization Relationship (GR) using the equation (iii).

$$GR = \sum(ni * wi) \quad \text{.... (iii)}$$

Where

w_i = the weight assigned to the generalization relationship

Dependency Relationship is relationship in which changes to one model (i.e. the supplier) impact another model (i.e. the client). It can exist in different cases:

i. Operation on A accesses a global variable of type B

ii. Operation on A has return value of type B

iii. Operation on A has parameter of type B

iv. Operation on A has a local variable of type B

We can calculate the Dependency Relationship (DR) using the equation (iv).

$$DR = \sum(ni * wg + ni * wr + ni * wp + ni * wl) \quad \text{.... (iv)}$$

Where

w_g = the weight assigned to the global variable relationship.

w_r = the weight assigned to the return type relationship.

w_p = the weight assigned to the parameter type relationship.

w_l = the weight assigned to the local variable relationship.

The number and type of connections or relationships among classes are appropriately taken into consideration in CSSM which was not present in Coupling between Objects (CBO), Response for a Class (RFC) and Depth inheritance Tree (DIT). This CSSM Metric gives a clear view of the internal structure which indirectly supports to evaluate the strength of coupling among the collaborating classes. The very essential design constraints on components are more cohesion and less coupling. If the coupling in a system is higher than the system is less reusable.

5.5 Theoretical Evaluation of Proposed Metric using Weyuker's Properties.

Weyuker [41] [42] proposed an axiomatic framework in the form of several properties for evaluating complexity aspects of software systems. These properties are:

Property 1: There exist P and Q, for which |P| ≠ |Q|, where P and Q are program units or components.

Property 2: Let C be a non-negative number, then there are finitely many program/components P for which |P|=C.

Property 3: There exist distinct components/program P and Q for which |P|=|Q|.

Property 4: There are functionally equivalent programs/components P and Q such that |P|≠|Q|.

Property 5: For any program/component bodies P and Q, we have |P| <= |P; Q| and |Q| <= |P; Q|.

Property 6: For program/component bodies P, Q and R such that |P|=|Q| and |P; R| ≠ |Q; R|.

Property 7: There are program/component bodies P and Q such that Q is formed by permuting the order of statements of P and |P|≠|Q|.

Property 8: If P is renaming of Q, then |P|=|Q|.

Property 9: There exist program/components bodies P and Q such that |P|+|Q|<|P; Q|.

These properties are evaluated for the proposed metric i.e. Coupling Based on Strength Specification Metric, as described below:

1) There may exist two different components/ program units which are having different coupling complexities, thus it satisfies the first property.
2) The coupling complexity cannot be negative because each component have at least one connectivity pattern with other component, thus the coupling complexity gives the positive value. It confirms the second property.
3) There may exist two different components with different functionality, but the proposed coupling complexity metric may have the same value for them, as there may have the same connectivity patterns between the components but providing different functionality. Thus it satisfies the third property.

4) There may be two components which are having same functionality but they may have different values of proposed coupling complexity metric, as these components may have different connectivity patterns and may be design using different concepts of technologies and programming. Thus it validates fourth property.
5) If one component is integrated with another component to get an assembled component for enhanced functionality, then the complexity of the individual components should be lesser then the complexity of the assembled component, as the connectivity between the two components increase the complexity of the integrated component. Thus it confirms fifth property.
6) Two components may have the same coupling complexity as they have the same connectivity patterns (i.e. UML relationship) between them. However they may be developed using different programming methodologies and therefore when these components are integrated with another component or in a system then they both have different integration code, relationship and implementation. Thus in both the cases it results in different coupling complexity value. Thus it validates sixth property.
7) If the order of parameters and methods is changed, it will not change the number of components need to be connected and the complexity caused by coupling methods, thus the proposed metric does not satisfies the seventh property.
8) The renaming of the parameters or components will not affect the complexity caused by the parameters or components. Thus it confirms the eighth property.
9) When two components are integrated or assemble it may results in new more connectivity patterns (i.e. dependencies) in order to provide the required functionality. Thus it validates the ninth property.

5.6 Empirical Evaluation of Proposed Coupling Metric Using A Case Study.

5.6.1 Case Study 1: To Show Difference between CBO and $CSSM_{CBO}$ Metrics

In order to empirically evaluate the proposed metric, we have considered a case study of Income Tax Department System. In this case study we have considered two components which are used to calculate the tax and pay the tax from different mode (that are Direct Payment and Payment through Agent). The class diagrams of these two components are

shown below in figure 5.4 and figure 5.5. In figure 5.4 the component show that how the employee and the employer can directly pay the tax online or manually and how they can calculate the tax. In figure 5.5 the component shows that how the employer and employee can indirectly (i.e. via agent) pay the tax manually only.

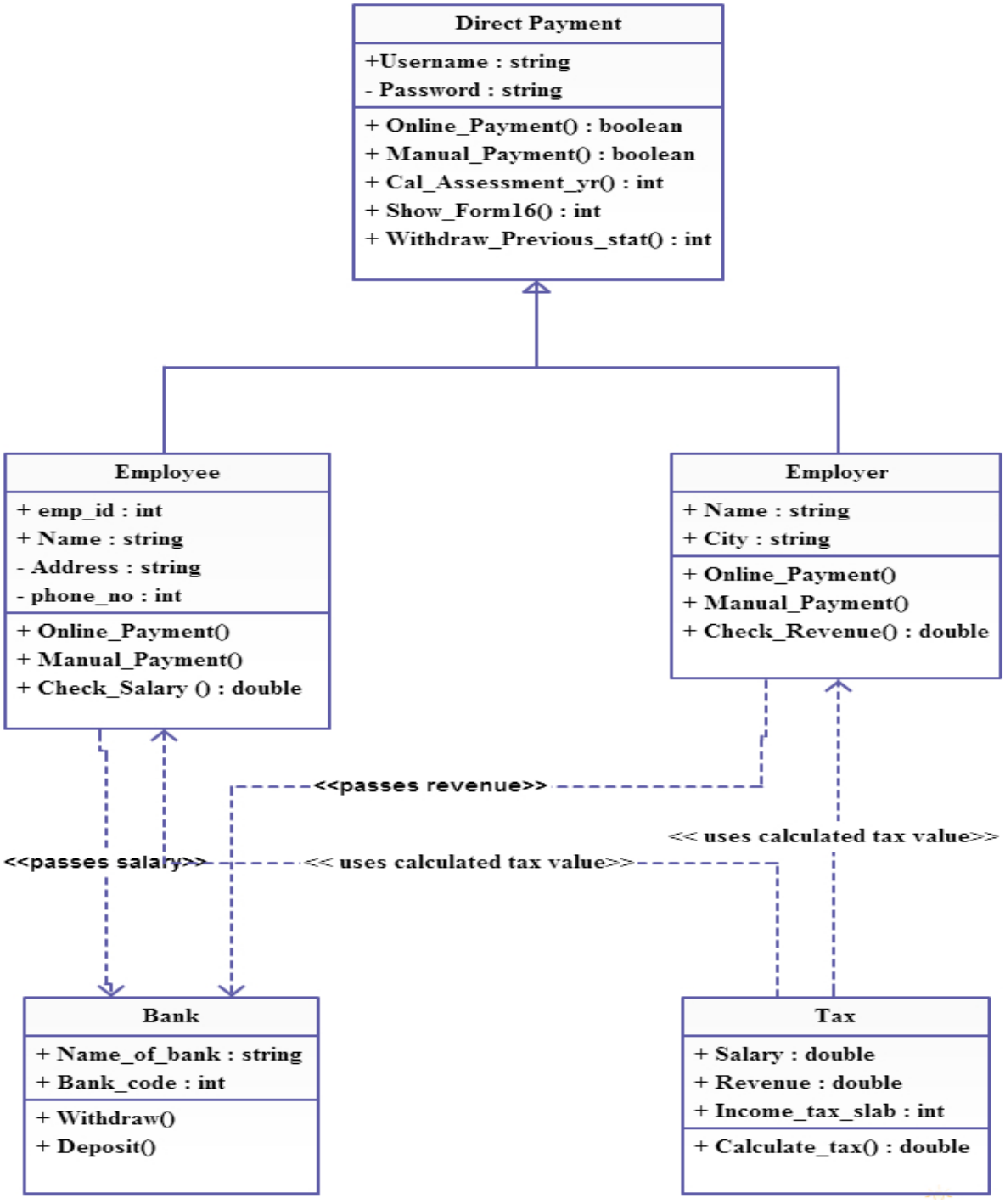

Figure 5.6: Class Diagram of Direct Payment of Income Tax Component

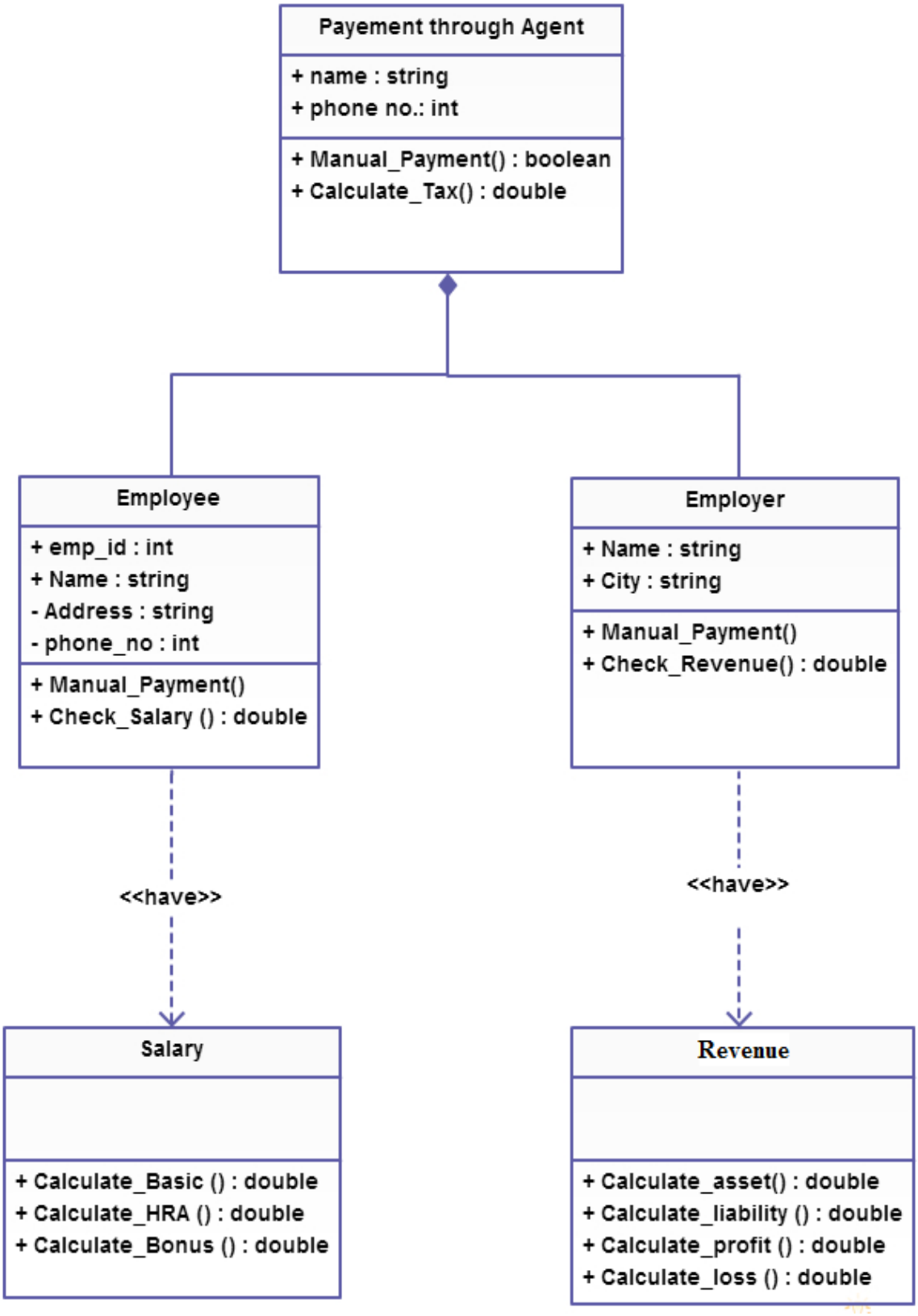

Figure 5.7: Class Diagram of Payment through Agent of Income Tax Component

A) Description of the Working of Different Components in the Income Tax Department System.

1) Direct Payment of Income Tax:

This component is composed of five different classes and their working has been described as below:

a) **Direct Payment:** This class checks the password entered by the user, in order to authenticate the user for the online payment of the tax. If the password is correct then it returns the value 1 otherwise 0. In this we also have the method of manual payment of the tax.

b) **Employee:** In this we contain the complete detail of the employee like emp_id, emp_name, Address, etc. The employee can check the salary using the Check_salary method which returns the salary in double value. This employee class inherits the member function of the Direct Payment class (that are Online_Payment and Manual_Payment)

c) **Employer:** This class contains the detail of the employer (i.e. different organization) like name, city, etc. It also inherits the member function of the Direct Payment class to pay the tax either manually or online. It also consists of its own method i.e. Check_Revenue() to check the revenue of the employer.

d) **Bank:** In this class we contain the name of the bank and the code of that particular bank. It consists of two member functions that are deposit and withdraw. The Employee and Employer classes pass the salary and revenue respectively as the parameter to the Bank class.

e) **Tax:** This class consists of salary, revenue and income_tax_slab attributes. In this class the tax is calculated using the tax member function and it returns the tax value (i.e. result) to the Employee and the Employer Class.

2) Payment through Agent:

This component is composed of five different classes and their working has been described as below:

a) **Payment through Agent:** This class contains two member functions that are manual_payment and calculate_tax. In this class an agent is present through

which the employee and employer pay the tax. In this we consist of the detail of the agent and he can calculate the tax of the employee and the employer.

b) **Employee:** The employee class contains all the details of particular employee and having member function to check his salary using the emp_id. This class has the composition relation with the Payment through Agent class; if Payment through Agent class is destroyed then the employee class is also destroyed. As in this the payment of tax is only done by the agent.

c) **Employer:** The employer class contains all the details of particular employer and having member function to check revenue. This class has the composition relation with the Payment through Agent class; if Payment through Agent class is destroyed then the employer class is also destroyed. As in this the payment of tax is only done by the agent.

d) **Salary:** In this class we can calculate the salary of employee using different methods like calculate_basic, calculate_HRA and calculate_bonus. When employee want to check_salary then the salary class passes the object of it as a parameter to the employee class which shows the parameter dependency between the employee class and the salary class..

e) **Revenue:** In this class we calculate the revenue of the employer using different member functions as shown in figure 5.5. When employee want to check_revenue then the revenue class passes the object of it as a parameter to the employer class. In this we have the parameter dependency between the employer class and the revenue class.

B) Steps to Calculate the Coupling between Objects (CBO) for Direct Payment Component:

1) Consider the component Direct Payment which is having five classes.
2) Apply the CBO metric on the components. It counts the number of classes that are coupled to a particular class i.e. where the methods of one class call the methods or access the variables of the other.
3) It gives the coupling complexity of direct payment component is equal to 2.

$$CBO_{\text{Direct Payment}} = 2$$

C) Steps to Calculate the Coupling Based on Strength Specification (CSSM) for Direct Payment Component:

1) Consider the component Direct Payment which is having 5 classes.
2) Apply the CSSM metric on the component:

2.a) when we apply the CSSM Metric to the component (i.e. Direct Payment) then we consider all the linkages or relationship between the classes. We have;

a) two generalization relationship
b) two return type relationship
c) two parameter type relationship

2.b) now put the weight assigned to the linkages from the table 5.2 in the equation (ii), (iii) and (iv) which gives the value of AR, GR, and DR.

$$GR = \sum(2 * 0.75)$$

$$= 1.50$$

$$DR = \sum 2 * 0.35 + 2 * 0.15$$

$$= 1.00$$

Put these value in equation (i)

$$CSSM = AR_c + GR_c + DR_c$$

$$= 0 + 1.50 + 1.00$$

$$= 2.50$$

D) Steps to Calculate the Coupling between Objects (CBO) for Payment through Agent Component:

1) Consider the component Payment through Agent which is having five classes with different member functions.
2) Apply the CBO metric on the component. It counts the number of classes that are coupled to a particular class i.e. where the methods of one class call the methods or access the variables of the other.
3) It gives the coupling complexity of Payment through Agent component is equal to 2.

$$CBO_{\text{Payment though Agent}} = 2$$

E) Steps to Calculate the Coupling Based on Strength Specification (CSSM) for Payment through Agent Component:

1) Consider the component Payment through Agent which is having 5 classes with different member functions.
2) Apply the CSSM metric on the component:

2.a) when we apply the CSSM Metric to the component (i.e. Payment through Agent) then we consider all the linkages or relationship between the classes. We have;

a) two composition relationship
b) two parameter type relationship

2.b) now put the weight assigned to the linkages from the table 5.2 in the equation (ii), (iii) and (iv) which gives the value of AR, GR, and DR.

$$AR = \sum(2 * 0.95)$$

$$= 1.90$$

$$DR = \sum 2 * 0.15$$

$$= 0.30$$

Put these value in equation (i)

$$CSSM = AR_c + GR_c + DR_c$$

$$= 1.90 + 0 + 0.30$$

$$= 2.20$$

F) Results

Table 5.9 shows the result of Case Study 2. In this it is clearly represented that the already existing Object Oriented Metric i.e. Coupling between Objects gives the ambiguous results. The newly proposed metrics give the unique results for same Case Study. It is because the already existing metrics does not consider the connectivity patterns or relationships which measuring the complexity of the component.

Table 5.9: Result of CBO and $CSSM_{CBO}$ Metric

Metric	Direct Payment Component	Payment through Agent Component
CBO	2	2
$CSSM_{CBO}$	2.50	2.20

5.6.2 Case Study 2: To Show Difference between RFC and $CSSM_{RFC}$ Metrics

In order to empirically evaluate the proposed metric w.r.t Response for a Class (RFC), we have considered two components which are consist of 3 classes. The class diagrams of these two components are shown below in figure 5.6 and figure 5.7. In figure 5.6 the component show that how the employee and the employer can directly pay the tax online or manually. In figure 5.7 the component shows that how the tax is calculated and information are passed between the classes.

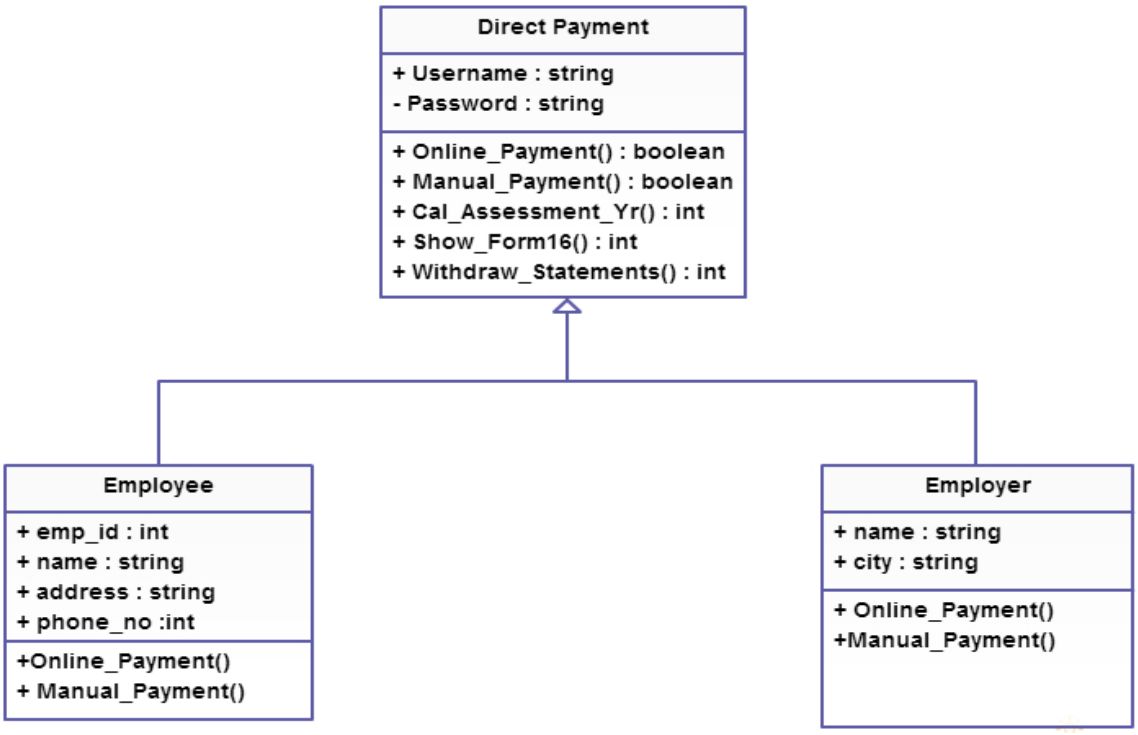

Figure 5.8: Direct Payment Component

In the figure 5.8 the direct payment component contains 3 classes with different member functions. The class employee and employer inherits the member function of the class Direct Payment. Using these inherited member functions the employee and employer can pay the tax either online or manually. This component shows the generalization relationship between the Direct Payment class and the Employee, Employer class.

A) Steps to Calculate the Response for a Class (RFC) for Direct Payment Component:

1) Consider the component Direct Payment which is having three classes and different methods in it.
2) Apply the RFC metric on the component. It counts all the methods that are called by the other methods in that class and all the methods in the class.
3) It gives the coupling complexity of direct payment component which is equal to 5.

$$RFC_{\text{Direct Payment}} = 5$$

B) **Steps to Calculate the Coupling Based on Strength Specification (CSSM) for Direct Payment Component:**

1) Consider the component Direct Payment which is having 3 classes.
2) Apply the CSSM metric on the component:

2.a) when we apply the CSSM Metric to the component (i.e. Direct Payment) then we consider all the linkages or relationship between the classes. We have;

a) two generalization relationship

2.b) now put the weight assigned to the linkages from the table 5.2 in the equation (ii), (iii) and (iv) which gives the value of AR, GR, and DR.

$GR = \sum(2 * 0.75)$

$= 1.50$

Put these value in equation (i)

$CSSM = AR_c + GR_c + DR_c$

$= 0 + 1.50 + 0$

$= 1.5$

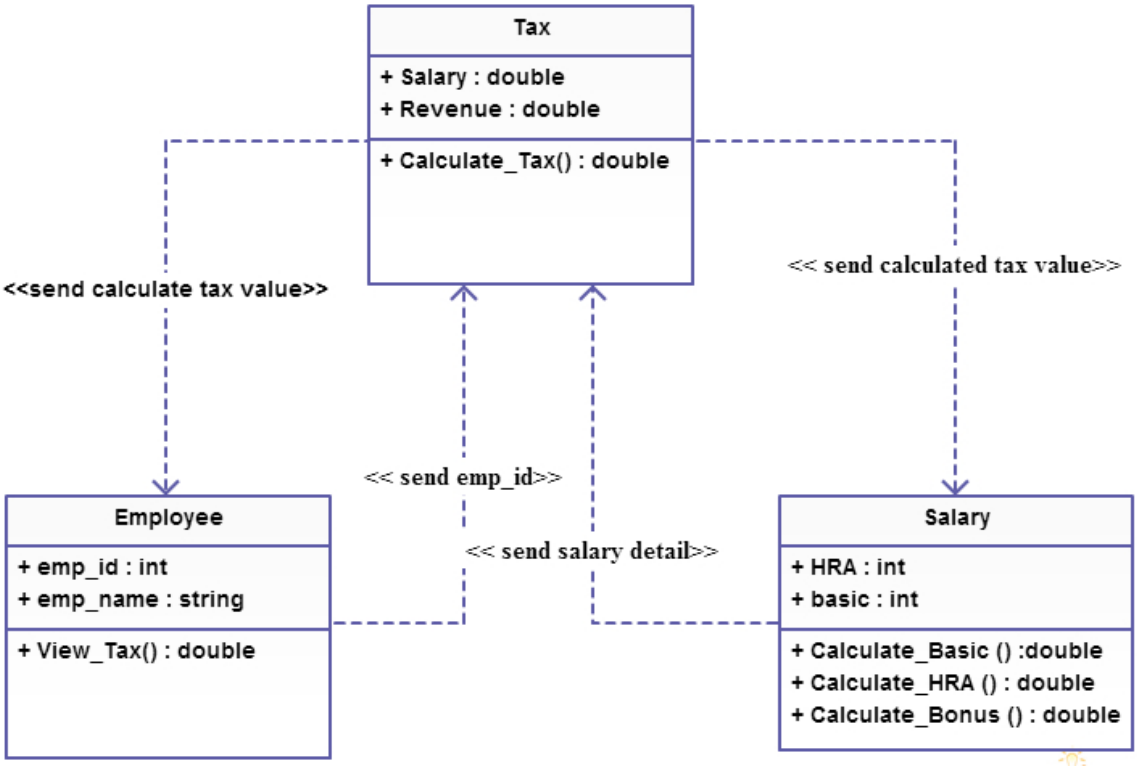

Figure 5.9: Tax Calculation Component

In the given figure 5.9 we have a component which consists of three classes with different number of member function in them. In this component there is dependency relationship between the methods (i.e. parameter dependency, return type dependency, etc). the employee passes the emp_id as the parameter to the calculate_tax method of the Tax class which is used to calculate the tax of particular employee. The calculate tax passes this id to salary class to get the salary detail of the employee. Now the salary class return the salary to the calculate_tax method and then calculate_tax send the value of tax after calculating to the employee to view the tax.

C) Steps to Calculate the RFC for Tax Calculation Component:

1) Consider the component Tax which is having three classes with different member functions.
2) Apply the RFC metric on the component. It counts all the methods that are called by the other methods in that class and all the methods in the class.
3) It gives the coupling complexity of tax component which is equal to 5.

$RFC_{Tax\ Calculation} = 5$

D) Steps to Calculate the Coupling Based on Strength Specification (CSSM) for Tax Calculation Component:

1) Consider the component Tax which is having three classes with different member functions.
2) Apply the CSSM metric on the component:

2.a) when we apply the CSSM Metric to the component (i.e. Tax Calculation) then we consider all the linkages or relationship between the classes. We have;

a) two parameter type relationship
b) two return type relationship

2.b) now put the weight assigned to the linkages from the table 5.2 in the equation (ii), (iii) and (iv) which gives the value of AR, GR, and DR.

$DR = \sum 2 * .35 + 2 * 0.15$

$= 1.00$

Put these value in equation (i)

$CSSM = AR_c + GR_c + DR_c$

$= 0 + 0 + 1.00 = 1.00$

E) Result

Table 5.10 shows the result of Case Study 2. In this it is clearly represented that the already existing Object Oriented Metric i.e. Response for a Class (RFC) gives the ambiguous results. The newly proposed metrics give the unique results for same Case Study. It is because the already existing metrics does not consider the connectivity patterns or relationships which measuring the complexity of the component.

Table 5.10: Result of RFC and $CSSM_{RFC}$ Metric

Metric	Direct Payment Component	Payment through Agent Component
RFC	5	5
$CSSM_{RFC}$	1.50	1.00

5.6.3 Case Study 3: To Show Difference between DIT and $CSSM_{DIT}$ Metrics

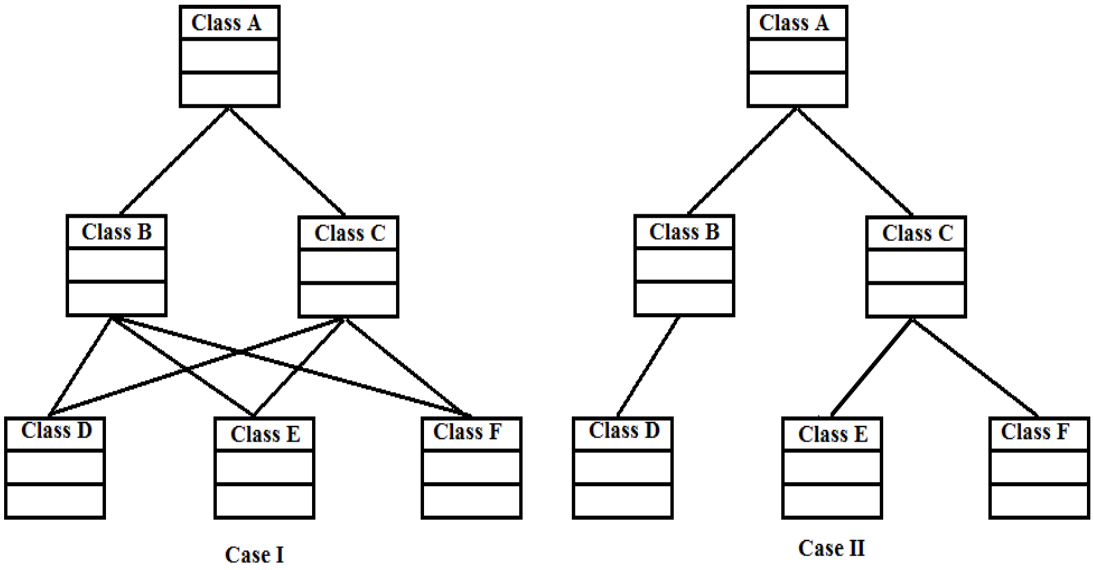

Figure 5.10: DIT Components

The figure 5.10 shows two components that Case I and Case II. In these cases all the the parent class is inherited by the child class, which shows the relations of generalization among the classes in a component.

A) Steps to Calculate the DIT for Case I:

1) Consider the component Case I which is having six classes with different member functions.
2) Apply the DIT metric on the component. It counts the classes that a particular class inherits from. It counts the level of subclasses affecting a class.
3) It gives the coupling complexity of Case I component which is equal to 2.

$$DIT_{Case\ I} = 2$$

B) Steps to Calculate the Coupling Based on Strength Specification (CSSM) for Case I.

1) Consider the component Case I which is having six classes with different member functions.
2) Apply the CSSM metric on the component:

2.a) when we apply the CSSM Metric to the component (i.e. Case I) then we consider all the linkages or relationship between the classes. We have;

a) Eight Generalization Relationship

2.b) now put the weight assigned to the linkages from the table 5.2 in the equation (ii), (iii) and (iv) which gives the value of AR, GR, and DR.

$$GR = \sum 8 * 0.75$$

$$= 6.0$$

Put these value in equation (i)

$$CSSM = AR_c + GR_c + DR_c$$

$$= 0 + 6.0 + 0$$

$$= 6.00$$

C) Steps to Calculate the DIT for Case II:

1) Consider the component Case II which is having six classes with different member functions.
2) Apply the DIT metric on the component. It counts the classes that a particular class inherits from. It counts the level of subclasses affecting a class.
3) It gives the coupling complexity of Case II component which is equal to 2.

$$DIT_{Case\ II} = 2$$

D) Steps to Calculate the Coupling Based on Strength Specification (CSSM) for Case II.

1) Consider the component Case II which is having six classes with different member functions.
2) Apply the CSSM metric on the component:

2.a) when we apply the CSSM Metric to the component (i.e. Case II) then we consider all the linkages or relationship between the classes. We have;

a) Four Generalization Relationship

2.b) now put the weight assigned to the linkages from the table 5.2 in the equation (ii), (iii) and (iv) which gives the value of AR, GR, and DR.

$$GR = \sum 4 * 0.75$$

$$= 3.0$$

Put these value in equation (i)

$$CSSM = AR_c + GR_c + DR_c$$

$$= 0 + 3.0 + 0$$

$$= 3.00$$

E) Result

Table 5.10 shows the result of Case Study 3. In this it is clearly represented that the already existing Object Oriented Metric i.e. Depth Inheritance Tree (DIT) gives the ambiguous results. The newly proposed metrics give the unique results for same Case Study. It is because the already existing metrics does not consider the connectivity patterns or relationships which measuring the complexity of the component.

Table 5.11: Result of DIT and $CSSM_{DIT}$ Metric

Metric	Case I	Case II
RFC	2	2
$CSSM_{RFC}$	6.0	3.0

5.7 Normalized Coupling based on Strength Specification ($CSSM_{Normalized}$)

5.7.1 Definition

Normalized Coupling based on Strength Specification ($CSSM_{Normalized}$) is defined as;

Ratio of the sum of weighted connections or relationships based on the strength of coupling existing among classes to the number of connections or relationships.

$$\text{CSSM}_{\text{Normalized}} = \frac{\sum_{i=0}^{n} Weighted\ n\ Types\ of\ Relationship}{Number\ of\ Relationship(n)}$$

Where, n is number of relationships.

$\text{CSSM}_{\text{Normalized}}$ value will lie in the range from 0 to 1. Values towards 0 will show simply coupled relationships, whereas values towards 1 show complexly coupled relationships. Coupling parameter is divided into three categories that are Complex, Medium and Low. These classifications are done on the basis of strength of coupling. The Change of strength of coupling parameter is considered to classify the parameters of coupling. The table 5.12 shows the coupling classification in three categories i.e. complex, medium and low.

Table 5.12: Coupling Classification

Rank	Coupling Parameters	Classification
RC1	Composition Relationship	Complex
RC2	Aggregation Relationship	
RC3	Generalization Relationship	Medium
RC4	Global Variable Dependency	
RC5	Return Dependency (Return object of other class)	
RC6	Return Dependency (Return Data Structure)	
RC7	Return Dependency (Return variable of other class)	Low
RC8	Parameter Dependency (Pass Parameter as Object)	
RC9	Parameter Dependency (Pass Parameter as Variable)	
RC10	Local Variable Dependency	

5.7.2 Perspective Probable Cases

In this the perspective probable cases are chosen on the basis of the type of relationships and the number of relationships.

i) Complex Case: The cases which are chosen are based upon the number of relationships which are defined above in table 5.12 under the complex type. Here all the perspective probable cases are taken which can lie under 0.75 to 1.00. The strength of coupling is very high under 0.75 to 1.00 range. In table below PC stands for perspective probable complex cases.

Table 5.13: Values for Probable Complex Cases

Relationship	**Weights**	**Number of Relationships(n)**							
		PC1	**PC2**	**PC3**	**PC4**	**PC5**	**PC6**	**PC7**	**PC8**
Composition	0.95	5	0	1	1	1	2	5	1
Aggregation	0.85	5	5	3	2	5	3	1	1
Results		.90	0.85	0.88	0.88	0.86	0.89	0.93	.90

ii) Medium Case: The cases which are chosen are based upon the number of relationships which are defined above in table 5.12 under the complex type. Here all the perspective probable cases are taken which can lie under 0.25 to 0.75. The strength of coupling is medium under 0.25 to 0.75 ranges. In table below PM stands for perspective probable medium cases.

Table 5.14: Values for Probable Medium Cases

Relationship	**Weights**	**No. of Relationships (n)**							
		PM1	**PM2**	**PM3**	**PM4**	**PM5**	**PM6**	**PM7**	**PM8**
Generalization	0.75	5	1	2	0	0	0	1	1
Global Variable	0.60	5	0	3	1	0	0	1	1
Return Type Dependency (Object)	0.50	5	0	1	1	3	1	2	1

Return Type Dependency (Data Stucture)	0.35	5	1	3	1	1	6	1	1
Results		0.55	0.54	0.53	0.48	0.44	0.37	0.54	0.55

iii) Low Case: The cases which are chosen are based upon the no of relationships which are defined above in table 5.12 under the complex type. Here all the perspective probable cases are taken which can lie under 0.00 to 0.25. The strength of coupling is low under 0.00 to 0.25 ranges. In table below PL stands for perspective probable Low cases.

Table 5.15: Values for Probable Low Cases

Relationship	**Weights**	**No. of Relationships (n)**							
		PL1	**PL2**	**PL3**	**PL4**	**PL5**	**PL6**	**PL7**	**PL8**
Return Type Dependency (Variable)	0.25	5	2	0	5	0	1	0	1
Parameter Dependency (Object)	0.15	5	3	0	4	2	0	1	1
Parameter Type Dependency (Variable)	0.10	5	4	1	3	1	0	1	1
Local Variable	0.05	5	5	2	2	0	3	2	1
Results		0.13	0.11	0.06	0.16	0.13	0.10	0.08	0.13

Chapter 6

Result and Analysis

Normalized Coupling based on Strength Specification (CSSM $_{Normalized}$) is defined as:

Ratio of the sum of weighted connections or relationships based on the strength of coupling existing among classes to the number of connections or relationships.

$$\text{CSSM}_{\text{Normalized}} = \frac{\sum_{i=0}^{n} Weighted\ n\ Types\ of\ Relationship}{Number\ of\ Relationship(n)}$$

Where, n is number of relationships.

Here the scale in figure 6.1 classifies the strength of coupling which lie in the ranges from 0 to 1. The range is divided into three categories i.e complex, medium and low.

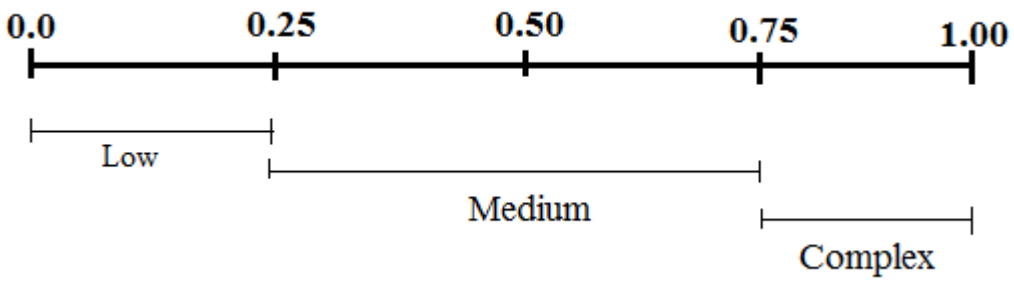

Figure 6.1: Classification of Strength of Coupling

Here three graphs are plotted for complex, medium and low cases. The X axis of the graph show number of relationships and the Y axis show the normalized CSSM values for the above three cases under consideration.

6.1 Plotting of Complex Cases: Normalized CSSM versus Number of Relationships

The graph below is plotted using the values from table no. 5.15.

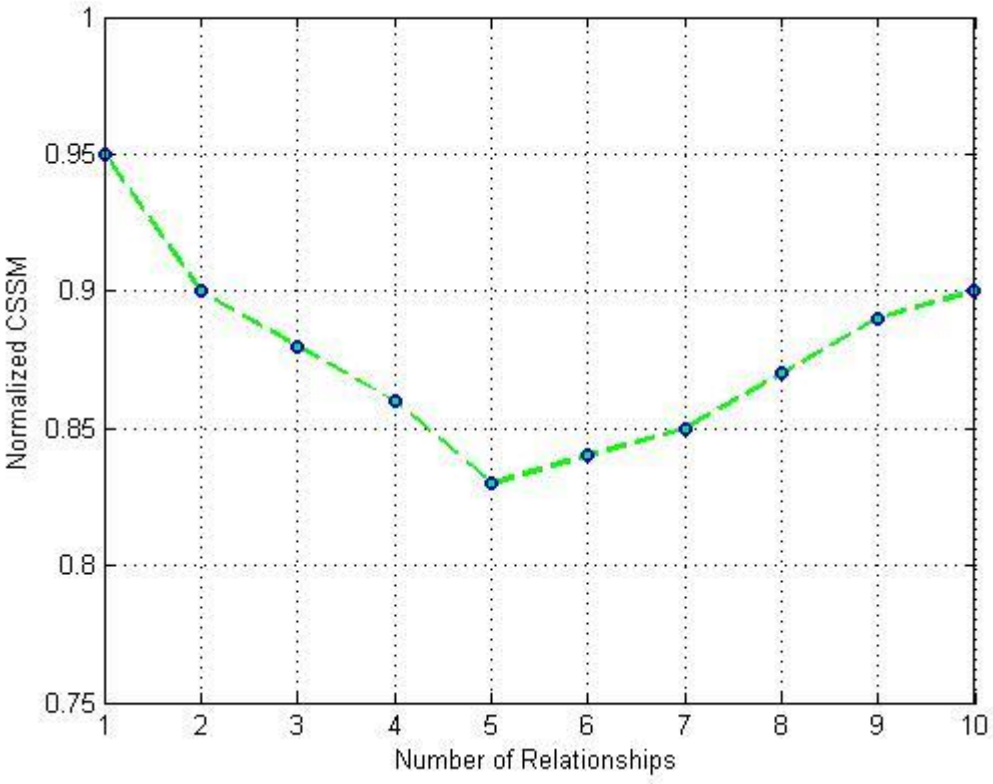

Figure 6.2: Plot for Complex Cases

In the Complex Case, strength of coupling lies between 0.75 to 0.95 ranges. The types of relationships considered are:

i) Composition Relationship

ii) Aggregation Relationship

The graph shows:

i) For lower number of relationship the Normalized CSSM value is higher.

ii) The Normalized CSSM firstly has the higher value for lower number of relationships and then it tends to decrease with increase in number of relationships.

iii) After optimum number of relationship the normalized CSSM will be increased.

This shows we need optimum level of coupling among classes. Neither lower number of connections nor higher numbers of connections is desired.

Inherent Property: number of relationship is directly proportional to number of classes. i.e. the lesser the number of relationship indicates that number of classes are also less. So in case of complex relationship if the numbers of classes are less, reusability strength of coupling considered to be high; because if two or three classes are there and complex relationships like composition and aggregation exist among these classes, it will increase the strength of coupling. And if number of relationship are higher that means number of classes are more and in more number of classes we can have complex relationship. Here again the CSSM will be on higher side. Number of classes and number of relationships needs to be kept neither very high nor very low for coupling.

6.2 Plotting of Medium Cases: Normalized CSSM versus Number of Relationships

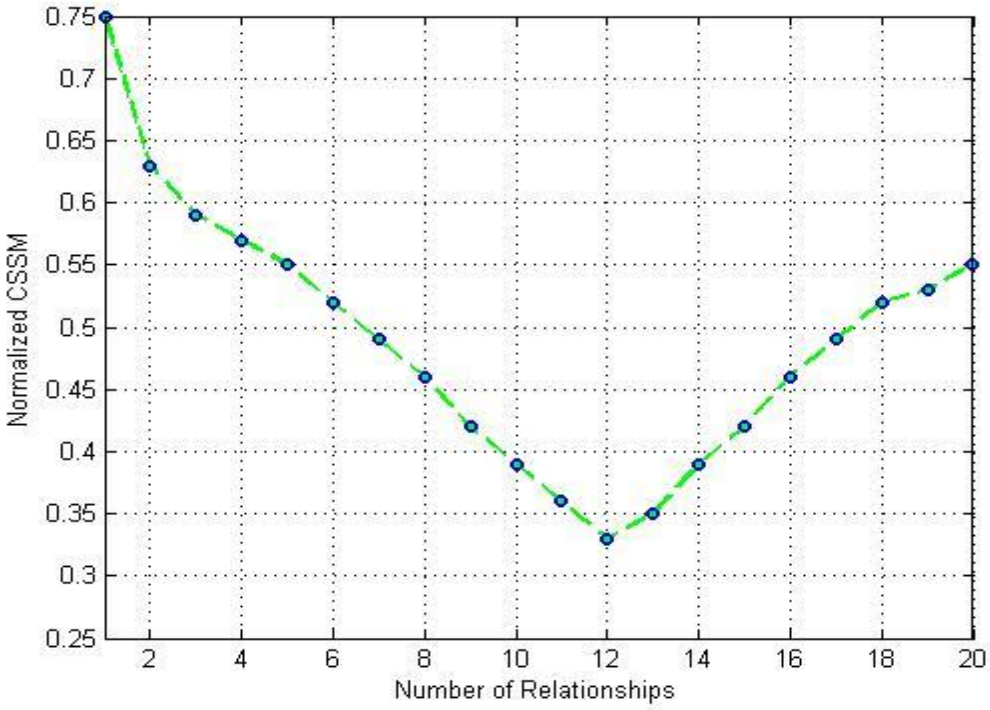

Figure 6.3: Plot for Medium Cases

Similar is the case with the Medium Case. In this the strength of coupling lies between 0.25 to 0.75 ranges. The types of relationships considered are:

i) Generalization

ii) Global Variable

iii) Return Type Dependency (Object)

iv) Return Type Dependency (Data Structure)

Normalized CSSM are on higher side when the number of relationship is very low and very high. It firstly decreases as the number of relationship increase and then after a certain point it will tend to increase with increasing number of relationships. The curve shows that optimum numbers of relationships are desirable for designing better software.

6.3 Plotting of Low Cases: Normalized CSSM versus Number of Relationships

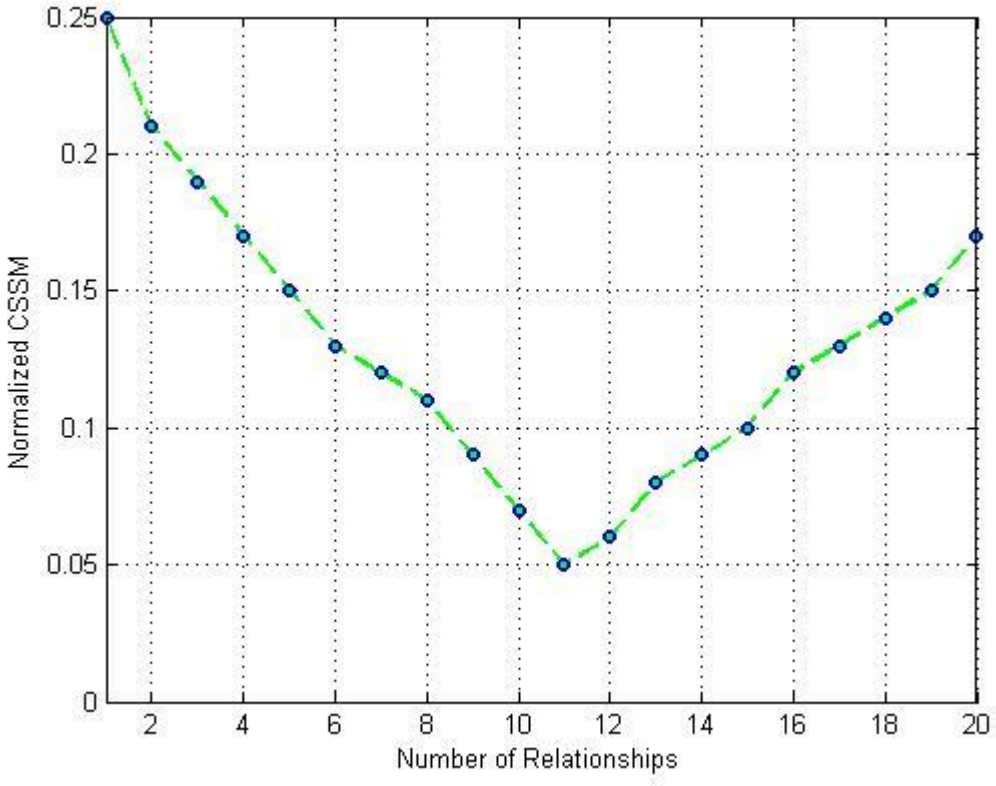

Figure 6.4: Plot for Low Cases

Similar is the case with the Low Case. In this the strength of coupling lies between 0.05 to 0.25 ranges. The types of relationships considered are:

i. Return Type Dependency (Variable)

ii. Parameter Type Dependency (Object)

iii. Parameter Type Dependency (Object)

iv. Local Variable

Normalized CSSM are on higher side when the number of relationship is very low and very high. It firstly decreases as the number of relationship increase and then after a certain point it will tend to increase with increasing number of relationships. The curve shows that optimum numbers of relationships are desirable for designing better software.

Chapter 7

Conclusion and Future Scope

7.1 Conclusion

The thesis introduces a metric to measure the coupling between classes on the basis of strength specification. Firstly, the Inability to Differentiate Anomaly (IDA) was identified and explained. The cases in which object oriented coupling metric like Coupling between Object (CBO), Depth Inheritance Tree (DIT) and Response for Class (RFC) suffered from the inability to differentiate anomaly were listed and discussed. Results showed that the Inability to Differentiate Anomaly (IDA) deserved attention and consideration. Then a metric is proposed i.e. Coupling based on Strength Specification (CSSM). This metric took into consideration the number of connections and the type of connections/ relationships. We compared the classical coupling metrics with the proposed metric and contrasted CBO and $CSSM_{CBO,}$ RFC and $CSSM_{RFC,}$ DIT and $CSSM_{DIT}$ and quoted the differences very clearly. And finally we gave Normalized CSSM which lie between the ranges 0 to 1. It classified the components according to their coupling complexity into three categories i.e. complex, medium and low. Number and type of connections among classes depict coupling or the strength of coupling and is an essential feature to be taken into consideration while designing the coupling metric. As CSSM is founded on the basis of number and type of connections, it proved to give more reliable results than the classical object oriented coupling metric like Coupling between Objects (CBO), Response for Class (RFC) and Depth Inheritance Tree (DIT).

7.2 Summary of Contributions

i. Identification of Inability to Differentiate Anomaly in object oriented coupling metric like Coupling between Objects, Response for Class and Depth Inheritance Tree.

ii. CSSM Metric is defined as the sum of weighted connections or relationships on the basis of strength of coupling existing among classes.

This CSSM metric take into consideration:

a) Number of Relationships
b) Types of Relationships
c) Number of Classes

This would be helpful in designing and coding. It also finds out coupling complexity of given software. It show the candidature of the component to interface it with other component on the basis of coupling complexity.

iii. Normalized CSSM Metric is defined as the ratio of the sum of weighted connections or relationships on the basis of strength of coupling existing among classes to the number of relationship.

7.3 Future Research

In the future

i) We can design an automated tool for calculating the coupling metrics on the basis of strength specification.

ii) The designing of reusable components can also done on the basis of CSSM metric.

iii) Interfaceability of two candidate components can also be calculated on the basis of CSSM.

References

[1] Ivica Crnkovic, Magnus Larsson, "Component-Based Software Engineering – New Paradigm of Software Development", http://scholar.google.co.in/scholar?q=ComponentBased+Software+Engineering+%E2%80%93+New+Paradigm+of+Software+Development.

[2] Parpanna Parthasarathy, "Component Integration Metrics and their Evaluation," Master Thesis and Specialist Projects, Western Kentucky University, 2007.

[3] Usha Kumari and Shuchita Upadhyaya, "An Interface Complexity Measure for Component-Based Software Systems," International Journal of Computer Applications, vol.36, no.1, pp.46-52, December 2011.

[4] Jarallah S. Alghamdi, "Measuring Software Coupling," The Arabian Journal for Science and Engineering, vol.33, pp. 119-129, April 2008.

[5] Usha Chhillar and Shuchita Bhasin, "A New Weighted Composite Complexity Measure for Object-Oriented Systems," International Journal of Information and Communication Technology Research, vol.1, pp. 101-108, July 2011.

[6] Bill Curtis, "Measurement and Experimentation in Software Engineering." In Proc. The IEEE, pp. 1144-1157, 1980.

[7] Roger S. Pressman, Software Engineering: A Practitioner's Approach, 6th ed., New York: McGrawHill,

[8] Ian Sommerville, "Software Engineering," 7th edition, 2004, Available at: http://ifs.host.cs.st-andrews.ac.uk/Books/SE7/Presentations/PDF/Ch19.pdf.

[9] Joakim Froberg, "Software Components and COTS in Software System Development"http://www.idt.mdh.se/cbse-book/extended reports/ 15_ Extended_ Report.pdf.

[10] J. Sametinger, "Software Engineering with Reusable Components", Available at: http://www.swe.uni-linz.ac.at/publications/abstract/TR-SE-97.04.html.

[11] K. Kaur and H. Singh, "Candidate Process Models for Component Based Software Development," Journal of Software Engineering, vol.4, pp.16-29, 2010.

[12] U. A. Khan, “The Evolution of Component Based Software Engineering from the Traditional Approach to Current Practices,” International Journal of Engineering and Management Research, vol.2, no.3, pp. 45-52, June 2012.

[13] Kung-Kiu Lau, Faris M. Taweel, and Cuong M. Tran, “The W Model for Component Based Software Development,” *in Proc. 37th EUROMICRO Conf on Software Engineering and Advanced Applications (SEAA)*, pp. 47-50, 2011.

[14] Ivica Crnkovic, Stig Larsson and Michel Chaudron, “ Component Based Developmnet Process and Component Lifecycle,” *in Proc. Int. Conf on Software Engineering Advances (ICSEA’06)*, 2006.

[15] Myers, G.J., Reliable Software Through Composite Design. Petrocelli / Charter,New York, 1975.

[16] Offutt, A.J., M.J. Harrold, and P. Kolte, A Software Metric System for Module coupling. Journal of Systems and Software, v.20: p. 295-308, March 1993.

[17] Page-Jones, M., The Practical Guide to Structured Systems Design. 2nd ed. Yourdon Press, New Jersey, 1988.

[18] Robert C. Martin:" Agile Software Development": Principles, Patterns and Practices, 2002.

[19] Booch, G., Rumbaugh, J., Jacobson, I. “The Unified Modeling Language User Guide”, Addison Wesley, Rational Software Corporation, 1999.

[20] Ivica Crnkovic, Magnus Larsson And Frank Luders, “The Different Aspects Of Component Based Software Engineering,”http: //citeseerx.ist .psu.edu/viewdoc/summary?doi=10.1.1.202.9388.

[21] Sheng Yu and Shijie Zhou, “A Survey on Metric of Software Complexity,” in *Proc. The 2nd IEEE Int. Conf on Information Management and Engineering* (ICIME), pp.352-356, April.16-18, 2010.

[22] Eun Sook Cho' and Min Sun fim2, “Component Metrics to Measure Component Quality” IEEE Software, pp 419-426, 2001.

[23] Tullio Vernazza, Giampiero Granatella, Giancarlo Succi, Luigi Benedicenti and Martin Mintchev, “Defining Metrics for Software Components,” in Proc. World Multiconfrence on Systemic, Cybernetics and Informatics, pp. 16-23, 2000.

[24] Hironori Washizaki, Hirokazu Yamamoto, “A Metrics Suite for Measuring Reusability of Software Components,” Ninth International Software Metrics Symposium (METRICS’03), 2003.

[25] Marcus A. S. Boxall, and Saeed Araban, “Interface Metrics for Reusability Analysis of Components,” in *proc*. Australian Software Engineering Conference (ASWEC’04), pp. 40-51, 2004.

[26] W.Stevens, G. Myers, and L. Constantine. Structured design. *IBM Systems Journal*,13(2):115 139, 1974.

[27] S. R. Chidamber and C. F. Kemerer. “A metrics suite for object oriented design.” IEEE Transactions on Software Engineering, 20(6):467-493, 1994.

[28] Briand, L. C., Devanbu, P., and Melo, W. L., "An investigation into coupling measures for C++", *in Proc. Of International Conference on Software engineering (ICSE'97),* Boston, MA, May 17 23 1997, pp. 412 – 421.

[29] Martin Hitz and Behzad Montazeri. “Measuring product attributes of object-oriented systems. In *Fifth European Software Engineering Conference*, pages 124 -136, Barcelona, Spain, September 1995.

[30] W. Li and S. Henry.” Maintenance metrics for the object-oriented paradigm.” *In Proceedings of the First International Software Metrics Symposium*, Baltimore Maryland, pages 52-60, May 1993.

[31] R. Harrison, S. Counsell, R. Nithi. “Coupling Metrics for Object-Oriented Design,” *IEEE Transactions on Software Engineering*, pages 1-8, 2006.

[32] Robert Martin, “OO Design Quality Metrics An Analysis of Dependencies,” *IEEE Transaction on Software Engineering*, pages 453-458, 2008.

[33] Haun Li.”A Novel Coupling Metric for Object Oriented Software System,” *IEEE Transaction on Software Engineering*, 2008.

[34] S.R.Chidamber and C.F.Kemerer, “Towards a Metrics Suite for Object Oriented DESIGN,” *Proc. Conf. Object Oriented Programming*: Systems, Languages and Applications, OOP- SLA’91, Oct 1991.

[35] N.I. Churcher and M.J. Shepperd, “Towards a Conceptual Framework for Object Oriented Software Metrics, “Software Eng. Notes, vol.20, no. 2, pp. 69-76, 2007.

[36] Sonia Chawla, Dr. Rajender Nath,"Evaluating Inheritance and Coupling Metrics", International Journal Of Engineering Trends and Technology (IJETT) - Volume 4 Issue 7 July2013.

[37] Imran Baig, "Measuring Cohesion and Coupling of Object Oriented System", IEEE Transactions on Software Engineering vo.14, pp. 254-259, August 2004.

[38] Aynur Abdurazik and Jeff Offutt, "Quantitatively Measuring Object-Oriented Couplings" In *Proceedings of the Eighth IEEE Symposium on Software Metrics (METRICS'02)*, pages 15- 27. IEEE, June 2002.

[39] Dirk Beyer, Claus Lewerentz and Frank Simon, "Impact of Inheritance on Metrics for Size, Coupling and Cohesion in Object Oriented Systems", IWSM 2000.

[40] Booch, G., Rumbaugh, J., Jacobson, I. "The Unified Modeling Language User Guide", Addison- Wesley, Rational Software Corporation, 1999.

[41] Elaine J. Weyuker, "Evaluating Software Complexity Measures," IEEE Transactions on Software Engineering, vol.14, no. 9, pp. 1357- 1365, September 1988

[42] Norman E. Fenton and Shari Lawrence Pfleeger, *Software Metrics: A Rigorous & Practical Approach*, 2nd ed., London: PWS Publishing, 1997.

[43] Lionel C. Briand, John W. Daly and Jaurgen K. Waust "A unified Framework for Coupling Measurement in Object Oriented Systems. IEEE Transactions on Software Engineering, 25- 91, January/ February 1999.

[44] L.C. Briand, J. Wust, J. Daly, and V. Porter, "Exploring the Relationship between Design Measures and Software Quality in Object-Oriented Systems," J. System and Software, vol. 51, no. 3, pp. 245-273, 2000.

Abbreviation

CBO	Coupling between Objects
CBS	Component Based Software
CBSE	Component Based Software Engineering
CBSD	Component Based Software Development
COTS	Commercial Off The Shelf
CSSM	Coupling based on Strength Specification
DIT	Depth Inheritance Tree
IDA	Inability to Differentiate Anomaly
LCOM	Lack of Cohesion Method
LOC	Line of Code
MAXDIT	Maximum of the DIT
MUT	Mean of Unrelated Trees
NOC	Number of Children
NOCC	Number of Children for a Component
OOP	Object Oriented Programming
RFC	Response for Class
RFCOM	Response Set for Component
WCC	Weighted Classes per Component

Publication

Published/ Accepted

1) Sangeeta Singh and Ashima Singh, "Measuring Components Integration: A Dilemma", IEEE Conference at Tunisia, May 2014.
2) Sangeeta Singh and Ashima Singh, "Component Based Software Metrics: An Insight", International Conference on Emerging Trends in Electrical, Electronics, Instrumentation & Computer Engineering (ETEICE), March, 2014

Communicated

1) Sangeeta Singh and Ashima Singh, "Object Oriented Coupling Metrics and Class Conectivity Patterns: Underlying Problems and Perspectives", IEEE Conference at Galgotias, Greater Noida, September, 2014.
2) Sangeeta Singh and Ashima Singh, "Identification of Ambiguous Cases in Object Oriented Coupling Metrics", IEEE Conference at JIIT, Noida, August 7-9, 2014.
3) Sangeeta Singh and Ashima Singh, "Issues and Challenges in Quantifying Modules and Components Integration", IEEE International Conference on Advances in Electronics, Computers and Communications at Reva, Bangalore, October 2014.