# Unified Class Coupling Model for Coupling Measurement in Object Oriented Software Systems

**Calvins Otieno**

**A thesis submitted in fulfillment for the requirements of the degree of Doctor of Philosophy in Information Technology in the Jomo Kenyatta University of Agriculture and Technology**

**2016**

## DECLARATION

This thesis is my original work and has not been presented for a degree in any other university.

Signature:  Date:……29/4/2016

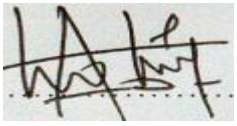
**Calvins Otieno**

This thesis has been submitted for examination with our approval as the university supervisor

Signature……:  Date:.....29/4/2016

**Dr Stephen Kimani**

**JKUAT,Kenya**

Signature:……  …… Date:…29/4/2016

**Dr. George Okeyo**

**JKUAT,Kenya**

## DEDICATION

I dedicate this thesis to the almighty God, loving father James Otieno Omware, mother Pamela Awinda Otieno for the great support and training and upbringing in my life and for support throughout my education.

## ACKNOWLEDGEMENT

My deep sincere gratitude to God the almighty and all those who supported me in the research. My acknowledgement to the management of School of Computing and Information Technology and the Chairman of Information Technology Department, Mr Philip Oyier. The two supervisors Dr Stephen Kimani and Dr George Okeyo for the supervision of this work. God bless you all and take good care of you.

## TABLE OF CONTENTS

## LIST OF TABLES

## LIST OF FIGURES

## LIST OF APPENDICES

## LIST OF ABBREVIATIONS

| | |
|---|---|
| **BSA** | Banking System Application |
| **Cam** | ClassessAccesing Methods |
| **CBO** | Coupling Between Objects |
| **CLC** | Class Level Coupling |
| **Ct** | Total Classes |
| **DAC** | Data Abstraction Class |
| **DLC** | Dynamic Level Coupling |
| **EOC** | Export Object Coupling |
| **ERS** | Empirical Relation System |
| **FMA** | Fashion Mobile Application |
| **H1** | Hypothesis One |
| **H2** | Hypothesis Two |
| **H3** | Hypothesis Three |
| $\mathbf{I_m}$ | Interacting Modules |
| **IOC** | Import Object Level Coupling |
| **JKUAT** | Jomo Kenyatta University of Agriculture and Technology |
| **LCOM** | Lack of Coupling in Methods |
| **LCOO** | Lack of Coupling in Operations |
| $\mathbf{M_a}$ | Methods Attributes |
| $\mathbf{M_h}$ | Methods Inherited |
| $\mathbf{M_i}$ | Methods Invoked |
| **Mp** | Methods Parameters |
| **N** | Number |
| **NAS** | Number of Associations |
| **NIC** | Number of Indirectly Connected Classes |
| **OLC** | Object Level Coupling |
| **OO** | Object Oriented |
| **OOP** | Object Oriented Programing |
| **OOS** | Object Oriented Systems |

| | |
|---|---|
| **RFC** | Response for Class |
| **RS** | Response Set |
| **SLC** | Static Level Coupling |
| **SMS** | Short Message Service |
| **$S_{rm}$** | Sum of Receiving Classess And Methods |
| **$S_t$** | Total Static Calls |
| **TCC** | Tight Class Coupling |
| **Tm** | Total Modules |
| **$U_{CC}$** | Unified Class Coupling |
| **UML** | Unified Modeling Language |
| **WOM** | Weighted Operations in Module |
| **DIT** | Depth of Inheritance Tree |
| **NOC** | Number of Children |
| **CAE** | Coupling on Advice Execution |
| **CMC** | Coupling on Method Calls |
| **CFA** | Coupling on Field Access |
| **RFM** | Response for Module |
| **CBM** | Coupling Between Modules |
| **WFC** | Weak Functional Coupling |
| **SFC** | Strong Functional Coupling |

## ABSTRACT

Coupling is the extent to which the functions performed by a subsystem are related. If a subcomponent is responsible for a number of unrelated functions then the functionality has been poorly distributed to subcomponents. Hence low coupling is a characteristic of a well designed subcomponent. The problem noted for research in this thesis is lack of standardization of measures, ambiguity in definition of measures and poor conceptual links between coupling components. The main goal of the research was to develop a model for coupling measurement in object oriented software systems.Literature review was conducted on coupling models and classification for coupling measures developed. In this research data was collected from various systems arising from students projects, analyzed in Poisson distribution model and the results used to formulate a unified coupling model. The model consist of five components that is class level coupling (CLC),object level coupling,static level coupling (SLC),dynamic coupling, and a combination of CLC and SLC. From the model a framework was developed for assessing coupling in software systems. To enhance the application of model and framework, an algorithm for assessing individual components and overall model was formulated and a tool to demonstrate model use created. The model was tested using five systems and results of high coupling and low coupling recorded.The model was validated using five softwares systems from github.com. From the validation three software recorded high coupling while two software systems recorded low coupling In conclusion the unified coupling model advances coupling measurement by providing ability to assess interrelation of all components and how they affect coupling behavior.This is an advancement of existing measurement models.

# CHAPTER ONE

## 1.0 INTRODUCTION

### 1.1 Research Overview

Coupling measures capture the degree of interaction and relationships among source codeelements, such as classes, methods, and attributes inobject-oriented (OO) software systems (Aryani *et al*, 2015). One of the main goals behind object oriented analysis and design is to implement asoftware system where classes have low coupling among them (Aryani *et al*, 2015). These class properties facilitate comprehension activities, testing efforts, reuse,and maintenance tasks (Ceret *et al*, 2010).Coupling is the extent to which the various components and subcomponents interact (Arlow & Neustadt, 2005). If various classes, methods and objects they are highly interdependent then changes to one is likely to have significant effects on the behavior of others (Dallal ,2007). Hence loose coupling between subcomponents such as classes, methods, objects and constructers is a desirable characteristic of an object oriented system (Ceret *et al*, 2010).Coupling can also be described as the extent to which the functions performed by a subsystem are related. If a subcomponent is responsible for a number of unrelated functions then the functionality has been poorly distributed to subcomponents. Hence low coupling is a characteristic of a well-designed subcomponent (Larman, 2005).

Coupling is an important property of software systems, which directly impacts program comprehension (Wilkie & Kitchenham, 2000). In addition, the strength of coupling measured between modules in software is often used as a predictor of external software quality attributes such as changeability, ripple effects of changes and fault-proneness (Kagdi,Gethers & Poshyvanyk, 2013).

During program comprehension, developers need to understand how software modules relate to each other (Sirbi & Kulkarani, 2010). It is especially important when

changes are being made to the software and developers need to assess the impact of their changes (Tonella, 2012). One way to understand such relationships is to measure the coupling between parts of the software. Coupling is one of the fundamental properties of software with a strong influence on comprehension and maintenance of large software systems (Bavota,Lucia & Oliveto, 2011).

Two different operationalization of technical coupling have been used in the large body of empirical work: syntactic dependencies and logical dependencies (Aryani & Lungu, 2014). Mortiz *et al* (2013) found that routines and modules with lower coupling were less likely to exhibit defects than those with higher levels of coupling. Analyses of object-oriented systems have reported similar findings when considering classes as the unit of analysis (Sirbi & Kulkarani, 2011). More recently, researchers have explored approaches to group software artifacts into various units of analysis and have examined the impact of coupling among those coarse-grained entities on the software quality (Genero, Piattini & Calero, 2011).

Coupling metrics capture the degree of interaction and relationships among source code elements in software systems (Kumar,Kumar & Rajesh, 2007). Coupling measures have important applications in software development and maintenance. They are used to reason about the structural complexity of software and have been shown to predict quality attributes such as fault-proneness, ripple effects of changes and changeability (Kumar,Kumar & Rajesh, 2007). Coupling measures are typically based on some form of static code analysis and their primary goal is to quantify the connectedness of a class to other classes (Stoy, 2013). For example, class A is coupled to class B if some method in A calls a method in B (Stoy, 2013). Coupling measures are used to help developers, testers and maintainers reason about software complexity and software quality attributes (Eddy *et al*, 2013). Coupling measures have been studied extensively and have been used to assist maintainers in tasks such as impact analysis, assessing the fault-proneness of classes, fault prediction, ripple effects, and changeability among others (Lossavio & Morantes, 2009). Therefore, coupling measures assist developers, testers and

maintainers in reasoning about the software and in predicting the needs for code inspection, testing and debugging (Krishnan *et al*, 2011) (Sirbi & Kulkarani, 2010). For object oriented software, the notion of coupling has not been considered with similar rigor by the pioneers who determined the major design guidelines of this paradigm (Gethers *et al* ,2012).There are two main reasons for this negligence (Gethers *et al* , 2012).:

- In structured design, there were few semantic guidelines to decompose a system into smaller subsystem. Consequently, syntactic aspects like size, coupling etc. played a major role. In contrast, in the object-oriented paradigm, the main criterion for systems decomposition is the mapping of objects of the problem domain into classes or subsystems in the analysis / design model, thus reducing the relative importance of syntactic criteria.
- Object-oriented analysis and design strive to incorporate data and related functionality into objects. This strategy in itself certainly reduces coupling between objects. Therefore, explicitly controlling coupling does not seem to be as important as in structured (especially top-down) design.

However, since employing object-oriented mechanisms in itself does not guarantee to really achieve minimum coupling, there is good reason to study coupling in object-oriented systems (Arora *et al*, 2012). In many cases, data or operations cannot be unambiguously assigned to one or another class on the grounds of semantic aspects, thus designers do need some kind of additional criteria for such assignments (Petrenko & Rajlich, 2009).Although introduction of classes is a powerful means for data abstraction it reduces the data flow between abstraction units and therefore reduces also total coupling within a system thus the number of variants of interdependency rises in comparison to conventional systems (Petrenko & Rajlich, 2009). This can be attributed to (Bidvi & Khare, 2012);

• The variety of mechanisms (inheritance, delegation, using- and has-relationships etc.) and

• The diversity of modules (classes and objects as well as functions and procedures in hybrid systems).

The different mechanisms can sometimes also be employed interchangeably, e.g., inheritance can sometimes be simulated by delegation, or using-relationships can sometimes be replaced by has-relationships etc (Bettini *et al*, 2010). Each of these variants exhibits different impacts on quality attributes which must be investigated and measured (Babu, 2015). The principles of encapsulation and data abstraction, although fundamental to object-orientation, may be violated to different extents via the underlying programming language (Arora,Singhall & Bansal, 2012). This leads to different strength of de-facto coupling which should be taken into account (John & Miller, 2012). In this spirit, several researchers have tried to adopt the notion of coupling for the object-oriented paradigm, Petrenko and Rajlich (2009) for instance, demands that objects from distinct classes should have as little coupling as possible, not only to make them more understandable, but so that they may easily be extracted from a particular application and reused in new situations. Thus, coupling seems to be even more important in object-oriented systems. For example coupling of client objects to a server object may introduce change dependencies (John & Miller, 2012).The tighter the coupling, the harder the effects on the clients whenever a crucial aspect of the server is being changed. High coupling between two objects makes it harder to understand one of them in isolation (John and Miller, 2012) . In contrast, low coupling leads to self-contained and thus easy to understand, maintainable objects ("KISS"-principle: "keep it simple & stupid") (Arora,Singhal & Bansal, 2012).

High coupling also increases the probability of remote effects, where errors in one object cause erroneous behavior of other objects. Again, lose coupling makes it easier to

track down a certain error, which in turn improves testability and eases debugging (Arora,Singhal & Bansal, 2012).

While creating complex application in java, the logic of one class will call the logic of another class to provide same service to the clients. If one class is calling the class logic of another class then it is called collaboration (Bavota,Lucia & Oliveto, 2011). When one class is collaborating with another class then there exists tight-coupling between the two classes (Bavota *et al* , 2015). If one class wants to call the logic of a second class then they first class need an object of second class it means the first class create an object of second class (Harter *et al*, 2011). For example, if we have two classes called traveller and car, traveller class is calling logic of car class; in this case traveller class creates an object of car class as shown in figure 1.1 below .The car class objects are of dependency for traveller object.

```
public class Car
{
  public void move(){}
}
public class Traveller
{
Car C=new Car();
public void startjourney()
{
c.move();}}
```

**Figure 1.1 Class Coupling**

In the above example traveller object is tightly coupled with car object because in place car object if you want to use C object then, we need to make changes in Car class

In Loose-Coupling, when one object is depending on another class object, some external entity will provide that dependency object to the main object that external object we call as a Container. In order to get loose-coupling between objects the following two rules are required

1. The classes should follow POJI/POJO model.
2. Apply dependency injection mechanism.

For example:-

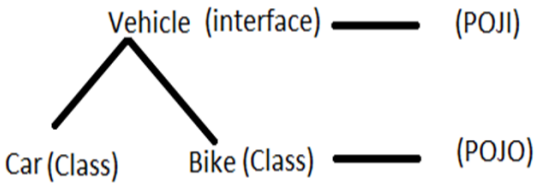



**Figure 1.2 Pojo Coupling Example**

```
public class Traveller{

Vehicle v;

public void set V(Vehicle v)

{

This V=V;

}

public void startjourney()

{

V.move();

}

}
```

**Figure 1.3 Loosely Coupled Classess**

In the above traveler class, an external entity injects either car (or) Bike object. In traveler, these are no changes required we are shifting the dependency from car to a Bike. In the above traveler class, we have token vehicle reference, so that an external object (Container) can injects either car object (or) Bike object, depends on requirement if a traveler. In spring frame work, spring container follows dependency injection mechanism and injects the dependency objects required for a main object. Spring frame work is much success because of one of the main reason;it promotes Loose-Coupling between the objects.

Many models have been proposed to measure the coupling and cohesion to predict the fault-proneness and maintainability of software systems (Lucia *et al*, 2008).

However, few studies have been done using coupling to measure reusability of software components because of their limitations and the difficulties to evaluate (Lucia *et al*, 2008).

Many maintenance tasks require the developer to measure directly or indirectly several attributes and assess properties of the software system under evolution. A variety of measures are proposed by researchers to assist developers in getting more complete views of the software (Aman *et al*, 2011).Coupling is one of the properties with most influence on maintenance as it has a direct effect on maintainability. Proposed coupling measures are used in tasks such as impact analysis (Larman, 2005), assessing the fault-proneness of classes (Briand *et al*, 2004), fault prediction (Briand et al, 2004), re-modularization, identifying of software components (Stein *et al*, 2009), design patterns (Larman, 2005) , assessing software quality (Aman *et al*, 2011), etc.

In general, one of the goals of the software designers is to keep the coupling in an OO system as low as possible (Marcus & Poshyvanyk, 2009). Classes of the system that are strongly coupled are most likely to be affected by changes and bugs from other classes; these classes tend to have an increased architectural importance and thus need to be identified (Henderson, 2011). Coupling measures help in such endeavors, and most of them are based on some form of dependency analysis, based on the available source code or design information.

Coupling is considered as one of most important OO software attributes. Many models have been proposed in the last several years to measure class coupling in object-oriented systems (OOS). Class coupling (more specifically, functional coupling) is defined as the degree of relatedness between members of a class (Pressman, 2005). In OOS, a class should represent a single logical concept, and not to be a collection of miscellaneous features. OO analysis and design methods promote a modular design by creating high coupled classes (Pressman, 2005). However, improper assignment of responsibilities in the design phase can produce low coupled classes with unrelated members.

The reasoning is that such (poorly designed) classes will be difficult to understand, to test and to maintain. However, there is no empirical evidence on these beliefs. In fact, studies have failed to show a significant relationship between, for example, coupling models and software quality attributes such as fault-proneness or changeability (Marcus & Poshyvanyk, 2009). Moreover, studies have noted that coupling models fail in many situations to properly reflect coupling of classes (Kabaili *et al*, 2010). One possible explanation of the lack of relationship between coupling and some software quality attributes is, according to some authors, due to the difficulty of measuring coupling from syntactic elements of code (Voas & Zhang, 2009).

In this thesis a model for assessing coupling in object oriented software systems has been developed.This thesis also presents a Model showing the sets of inputs and outputs.This thesis also presents an algorithm for assessing unified coupling in object oriented software systems.This thesis also presents a tool automating the developed model in this research.

### 1.2 Problem statement

Coupling is a more complex software attribute in object oriented systems but our understanding about coupling measurement factors is poor (Gelinas *et al*, 2011). There is no standardization for expressing coupling measures; many measures are not operationally defined i.e. there is some ambiguity in their definitions (Gelinas *et al*, 2011). As a result, it is difficult to understand how different measures relate to one other and what their potential use is. All the above aspects ultimately shapes the need of detailed study of coupling measurement in object-oriented systems.

Briand *et al* (2004) in their research notes that there is little understanding of the motivation and empirical hypotheses behind many of these new measures (Briand *et al*, 2004). It is often difficult to determine how such measures relate to one another and for which application they can be used.

As a consequence, it is very difficult for practitioners and researchers to obtain a clear picture of the state-of-the-art in order to select or define measures for object-oriented systems(Briand *et al*, 2004).

The fact that there also exists little empirical validation of existing object-oriented coupling measures means the usefulness of most measures is not supported by strong industry results to validate their accuracy (Gelinas *et al*, 2011).

A vast majority of coupling models abound in the literature relies on structural information, which captures relations, such as method calls or attributesusages. However, these structuralmodels lack the ability to identify conceptual links, which, for example, specify implicit relationships encoded in identifiers and objects in source code

## 1.3 Research Objectives

### 1.3.1 Main Objective

The main objective of this research was to develop a unified class coupling model for evaluating coupling levels in object oriented software systems. To achieve this the following specific objectives guided the study:-

### 1.3.2 Specific Objectives

To achieve the main objectives in this research, we intended to undertake the following:

1. To review coupling models and develop a classification scheme.
2. To analyze data collected from software developers and various software systems with regards to software coupling.
3. To develop unified class model for coupling measurement in object oriented software systems.

4. To assess the effectiveness of the proposed unified class models with existing object oriented software systems.

## 1.4 Research Questions

1. How can the current models be classified in accordance to coupling measurements?
2. What are some of the automated tools developed to assess software coupling in object oriented systems.
3. What are the weaknesses of existing software models for assessment of software couplings that can be addressed by the proposed unified model in this research?
4. How can existing models be enhanced to generate a unified class model?
5. How does the proposed models compares with other existing models for coupling measurement?

## 1.5 Proposed Solutions

To address and clarify our understanding of the state-of the-art of coupling measurement in object-oriented systems, this research proposed to develop amodel that:

1. facilitate comparison of existing measures,
2. facilitate the evaluation and empirical validation of existing measures
3. Support the definition of new measures and the selection of existing ones based on a particular goal of measurement.

This is achieved through the following knowledge contributions:-

1. Development of amodel for assessment of the various software systems
2. Development of a framework describing the set of inputs and outputs for the model has been proposed for assessment of software systems

3. Development of an algorithm to assess individual components of the software systems and an overall algorithm for assessment of the overall coupling in the software systems
4. Development of a tool in Java program to help users interact easily and assess the various elements of software system. The tool implemented both the algorithms and mathematical evaluations proposed in the model.

### 1.6 Scope

1. A research was conducted on software coupling in object oriented software systems.
2. Literature review was conducted on coupling models and measures
3. Data was collected from software systems developed in JKUAT university to allow for ease of access of the software systems and reduce on time and proprietary challenges.
4. Data collected was based on coupled measures with regards to lines of codes in software systems
5. The research did not divulge on software maintenance effort although it is one of the factors affected by coupling in object oriented software systems.
6. The project proposed a unified coupling model for measurement of software coupling
7. A model and algorithm was developed for coupling measurement.

### 1.7 Study Limitations

This study majorly focused on the software industry and individuals who design and develop software systems for organization. The research work encountered various challenges such as:-

1. Accessing all respondents due to time limit factor.

2. Deep knowledge of coding and designing of software systems was required which limited the number of respondents from the software systems collected.
3. Mistrust from the various software developers due to proprietary of their work and there assumptions that the researchers wanted to steal their codes.
4. Unfamiliarity on coupling models applicable in use in assessing software systems on part of the software developers and the research team.
5. Inadequate reference journal from the university library and online repository.

# CHAPTER TWO

# 2.0 LITERATURE REVIEW

## 2.1 Introduction

Coupling measurement is a focus of study for many of the software professionals for the last few years. Object-oriented programming is an efficient programming technique for programmers because of its features like reusability, data abstraction, inheritance and method overloading (Bidve & Khare, 2012). Coupling is a very important factor in object-oriented programming for software quality measurement and used as predictors of software quality attributes such as fault proneness, impact analysis, ripple effects of changes and changeability (Bidve & Khare, 2012).

Class coupling (more specifically, functional coupling) is defined as the degree of relatedness between members of a class (Pressman, 2005). In object oriented software systems, a class should represent a single logical concept, and not to be a collection of miscellaneous features (Poshyvanyk *et al*,2009). Object oriented analysis and design methods promote a modular design by creating lowly coupled classes (Pressman, 2005).

One of the main goals behind Object Oriented analysis and design is to implement a software system where classes have low coupling among them (Petrenko *et al* ,2007). These class properties facilitate comprehension activities, testing efforts, reuse, and maintenance tasks (Dennis,Wixom & Tegarden, 2015).

Coupling is the extent to which the various components and subcomponents interact. If various classes, methods and objects are highly interdependent then changes to one are likely to have significant effects on the behavior of others. Hence loose coupling between subcomponents such as classes, methods, objects and constructs is a desirable characteristic of an object oriented system (Dennis,Wixom & Tegarden, 2015).

In this research coupling will be defined as the extent to which the various components and subcomponents interact. If various classes, methods and objects are highly interdependent then changes to one are likely to have significant effects on the behavior of others (Dennis,Wixom & Tegarden, 2015).

To evaluate and maintain quality of object-oriented software there is a need to assess and analyse its design and implementation using appropriate measurement model (Jungmayr, 2002). A quality model should relate to external quality attributes of a design. External quality attributes include maintainability, reusability, error proneness, and understandability (Voas & Zhang, 2009).

In general, one of the goals of software designers is to keep the coupling in object-oriented system as low as possible ( Kazemi *et al*, 2011). Classes of the system that are strongly coupled are most likely to be affected by changes and bugs from other classes. Such classes have more architectural importance; coupling measures helps in such happenings (Jungmayr, 2002).

## 2.2 Existing Coupling Measures

### 2.2.1 Package Coupling Measures

For object-oriented systems, most of the coupling model have been defined up to class level and only a few models exist for measurement of coupling at the higher levels of abstraction (Hongyu *et al,* 2010). Other work related to packages or other higher abstraction levels has been carried out (Hongyu *et al,* 2010). Liu et al proposed information-based coupling (ICP), a coupling measure for a set of classes based on information flow through method invocations within classes(Liu *et al*,2009). They defined coupling of a set of classes as the sum of the couplings of the classes in the set (Hongyu *et al,* 2010). Martin proposed various package level coupling measures such as package coupling, They defined package coupling as the number of classes from other

packages that depend on the classes within the package (Genero,Piattini & Calero,2005).He also defined independent package coupling as the number of classes from other packages that the classes within the package depend upon (Farias,Garcia & Lucena, 2012). In addition, he defined abstractness as the ratio of abstract classes to the total number of classes and instability as the ratio of package coupling to total coupling (France *et al*, 2006). Similarly, Tagoug proposed a coupling measure for subjects (Bird *et al*, 2009). The concept of subjects is quite similar to the packages and he defined subject coupling based on interactions among subjects and also proposed a decomposition quality model for object-oriented systems on the basis of difference between cohesion and couplings of elements of a system (Bird *et al*, 2009). Tagoug did not consider the hierarchy of subjects while defining above measures. Also, no justification packages in an object-oriented system have hierarchical structure and each package can be viewed as a set of elements (classes or sub-packages) and relationships between these elements (Bird *et al*, 2009).

These relationships among elements of different packages present at the same hierarchical level give rise to the inter-package coupling (Perepletchikov *et al*,2007). In other words, two different packages present at the same hierarchical level are said to be coupled if the elements of these packages are connected to each other (Poshyvanyk *et al*, 2009). The elements of the two different packages are said to be connected if there exists a relationship between them and this relationship can be of the type of inheritance relationship, aggregation relationship or simple reference relationship (Burrows *et al*, 2010).

This connection or relationship between elements of two different packages is denoted by $r(e_i; e_j)$. If there is a connection between two elements $e_i$ and ej , then, $r(e_i; e_j) = 1$. The presence of connection between two elements can also be represented by $e_i ; e_j$ . Such types of connections always have a direction (Qusef *et al*, 2011) i.e., if element $e_i$ is connected to element $e_j$ , then it is not necessary that element $e_j$ is also connected to element $e_i$ .Because an element may be dependent on another element, the reverse of this

may not be true (Qusef *et al*, 2011). For example, if we assume that an element $e_1$ is dependent on $e_2$, then it means that any change in implementation of element $e_2$ is likely to initiate some changes in implementation of element $e_1$ (Qusef *et al*, 2011).

But any change in implementation of element $e_1$ will not necessarily demand changes in implementation of $e_2$. Thus, connection between a pair of elements has to be considered as a directional entity. Thus, connection between a pair of elements is asymmetric model in nature,i.e., $r(e_i; e_j) \neq r(e_j ; e_i)$ (Yang *et al*, 2015). This important aspect of direction of coupling has also been considered at package level in this work (Yang *et al* , 2015). The elements of a package may be classes, interfaces or sub-packages. However, interfaces are unique to C# and Java languages and are quite similar to abstract classes in structure as well as in behavior, as far as inter-package coupling measurement is concerned (Yang *et al*, 2015).

The research carried out identified the following coupling connections

1. Class-Class Connection: If both concerned elements of packages are classes (or interfaces), then there exists a class-class type of connection between them (Zhao *et al* , 2016). These class elements of two packages are said to be connected to each other, if any one of the following relationships exists between them (Zhao *et al* , 2016):

(i) Two classes are related by aggregation relationship,i.e., one class has the other class as type of one of its attribute;
(ii) One class inherits another class or one class implements an interface;
(iii) One class has got a method that is invoking method of another class;
(iv) One class has a method referencing an attribute of another class;
(v) One class has got a method having parameter of the type of another class;
(vi) One class has a method containing a local variable of the type of another class;
(vii) One class has a method invoking a method having a parameter of the type of another class.

2. Sub-Package-Sub-Package Connection: Packages may consist of sub-packages as its elements at the next level (Yang *et al*,2015). Thus, while measuring inter-package coupling, we have to consider connections between pairs of such elements of packages. Such types of connections are recursively defined, as one sub-package is said to be connected to another sub-package if elements of one sub-package have got one or more relationships with the elements of another sub-package (Zhao *et al* , 2016).

### 2.2.2 Class Coupling Measures

In this section we reviewed materials relating to class coupling measure from previous research work.

Hitz and Montazeri, (2011) described Class level coupling (CLC) as coupling resulting from state dependencies between two classes in a system during the development lifecycle. According to Hitz and Montazeri (2011), class level coupling is important when considering maintenance and change dependencies because changes in one class may lead to changes in other classes which use it (Yadav & Khan, 2012). The authors also state that CLC can occur if a method of a class invokes a method or references an attribute of another class. For example if we, let cc be the accessing class (client class), scbe the accessed class (server class). The factors determining the strength of CLC between client class c and server class are (Zhou *et al*,2014):

1. Stability of sever class: sever class is stable: Interface or body of sc is unlikely to be changed (for instance due to changing requirements). Typically, basic types provided by the programming language, or classes imported from standard libraries are stable.
2. Scope of access: Determines where sc is visible within the definition of cc. Within this scope, a change to sc may have an impact on cc. The larger the scope, the stronger the classes are coupled.

Bieman and Kang (1999) proposed two class coupling measures to evaluate the relationship between class coupling and private reuse in the system (Bieman & Kang, 1999).

Bieman and Kang research focuses on two important criteria of coupling measure;

1 the interaction pattern
2 The special method.

Bieman and Kang (1999) defined class coupling as an attribute of a class that refers to the connectivity among components of the class. The components include the instance variables, methods defined in a class and methods that are inherited (Bieman & Kang, 1999).

According to Bieman and Kang (1999), a method is said to be connected to an instance variable if the method accesses that instance variable. However, the manner in which an instance variable may be used is different. Two methods that access one or more common instance variables are said to be directly connected, while two methods that are connected through other directly or indirectly connected methods are indirectly connected (Bieman & Kang, 1999).

According to Biemen and Kang (1999) we let ndc (c) be the number of directly connected methods in a class c, nic (c) is the number of indirectly connected methods in a class, and np (c) is the maximum possible number of connections in a class. Then, tight class coupling (tcc) is defined as (Bieman & Kang, 1999)

**tcc(c) = ndc(c) / np(c) ……*Equation 2.1***

and loose class coupling (lcc) is defined as:

**lcc(c) = (ndc(c) + nic(c)) / np(c) …..*Equation 2.2***

According to Bieman and Kang loose coupling exist for values between 0.0 to 0.499 while strong coupling exist for values between 0.5 to 1 (Bieman & Kang, 1999). This model focuses on two important criteria of coupling measure; the interaction pattern and the special method. The strongest part in this model is to consider the interaction between pattern methods (Zhou,Yang & Leung ,2009). However, the connectivity factor could generate misleading information leading to poor recognition of the interaction pattern (Voas & Zhang, 2009).

Access methods also cause problems for measures that count pairs of methods that use common instance variables. Because access methods usually access only one instance variable, many pairs of methods that do not reference a common instance variable can be formed using access methods (Voas & Zhang, 2009). Thus, the coupling is artificially decreased. The best solution to pattern recognition problem is to exclude access methods from analysis. Similarly for constructor which typically reference all instance variables, pattern recognition which artificially increases the coupling of the class, this model is not suitable because it generates many pairs of methods that use an instance variable in common. It is therefore better to exclude constructors from analysis (Breivold,Crnkovic & Larsson, 2012).

In another research,Chae *et al* (2004) considers two characteristics of classes, which could lead to different coupling values from the general intuition of coupling (Chae *et al*, 2004). The two characteristics are: the patterns of the interactions among the class members (e.g. counting the number of instance variables used by a method) and special methods. According to them the nature of these methods should be reflected to properly measure the coupledness (Chae *et al*, 2004).

According to Chae *et al* (2004), in order to describe these characteristics in class c, a reference graph $g_r(c)$ is drawn to represent the interaction among the members of class c, and is defined to be an undirected graph $g_r=(n,a)$ with (Chae et al ,2010): The

relationship for evaluating coupling in the models is given as follows (Counsell *et al*, 2006)

$$\boldsymbol{N = V(C) \cup M(C)} \quad \textbf{……} \quad \textbf{\textit{Equation 2.3}}$$

$$\boldsymbol{A = \{(M, V)!\, V \in R(M)\}} \quad \textbf{……} \textbf{\textit{Equation 2.4}}$$

Where V(C) is a set of instance variables in the class c, M(C) is a set of methods in the class c, and $R^{*}$(M) is a set of instance variables directly/indirectly references by M.

In their research Chae *et al* (2004) define glue methods of graph $g_r$, $m_g(g_r)$, as the minimum set of methods without which its reference graph $g_r$ can be divided into disjoint sub-reference graphs (Cox *et al*,2006). Then, they introduce the notion of connectivity factor as: the connectivity factor of a reference graph $g_r$, $f_c(g_r)$, represents the degree of the connectivity among the members, and is defined to be the ratio of the number of the glue methods to the number of normal methods (Chae *et al* ,2004).

Briand *et al* (2000) define four coupling properties to characterize coupling in a reasonable intuitive and rigorous manner. The four properties are: nonnegative and normalization, null value and maximum value, monotonicity, and merging of unconnected classes (Briand *et al*, 2000). However, these properties are not sufficient to tell that a measure that fulfils them all will be useful; it is likely that a measure that does not fulfill them all is ill-defined. Let $R_c$the set of relationships within the class c. The set of all intra-class relationships in an object-oriented system is defined as $R_c$. We say that $R_c$is maximal, if all possible relationships within class c are presented. We say $R_c$ is maximal if $R_c$ is maximal $\forall c \in C$ (Briand *et al*, 2000).

In the very first research conducted by Briand *et al* in 2000, it was noted that we must have a consensus on the terminology and formal definition of measure expression in software engineering before we are really able to propose and apply a model. On the

other hand, it is stated implicitly that we have to validate the models that we are going to use for assessing or quantifying the attributes of software (Briand *et al*, 2000). Briand et al added some criteria to compare various approaches in order to be in the same perspective (Briand *et al*, 2000). The five criteria of the model to address these varieties are: types of connection (what makes a class coupled), domain of the measures, direct and indirect connections, inheritance, and access methods and constructors (Lucia, Oliveto & Vorraro, 2008).

A research conducted by Harrison *et al* (2000) defines coupling between classes (CBC) as a count of the number of classes to which a class is coupled. It counts each usage as a separate occurrence of coupling. This includes coupling via inheritance. This approach only performs normal count and concludes that coupling exists if count is greater than zero (Harrison *et al*, 2000).

The research done by Harrison *et al* (2000) considers the Number of Associations (NAS) in a class. In this research three hypotheses related to coupling are investigated by the authors are (Harrison *et al*, 2000):

1. $\mu_1$: As inter-class coupling increases, the understandability of a class decreases. This hypothesis is rejected by authors.
2. $\mu_2$: As inter-class coupling increases, the total number of errors found in a class increases. This hypothesis is rejected by authors.
3. $\mu_3$: As inter-class coupling increases, the error density of a class increases. This hypothesis is supported by authors.

To investigate these hypotheses author studied dependent variables such as

1. Software Understandability (SU)
2. Number of known errors (KE)
3. Error per thousand non-comment source lines (KE/KNCSL)

Also, Coupling due to object as a parameter of methods and return type for a method is considered by other authors Dang and Zhang (2005).In this research it is noted that source lines of codes model measures the number of physical lines of active code, that is, no blank or commented lines code (Conchuir *et al* ,2009). Counting the source lines of code is one of the earliest and easiest approaches to measuring complexity. In general the higher the Source lines of code in a module the less understandable and maintainable the module is (Conchuir *et al* ,2009).

In another research as proposed by Simon *et al* (2008), they advances one major principle in software engineering; design principle, "put together what belongs together".Three major criteria (point of views) that might lead this grouping are functionality, data orientation, and time orientation. The grouping process, which is called the extraction for reverse engineering, should be tool supported. One instrument for their implementation is distance measure (Simon *et al*, 2008).

In the research the work of groupings is strongly connected with the theory of similarity/dissimilarity. Simon *et al* (2008) proposes that the degree of similarity in coupling measurement can be defined quantitatively if all properties of the object are assigned the same weight (uniform). In this research is proposed that the degree of similarity between two things x and y relative to a finite subset b of all properties $p_i$ is

$$\boldsymbol{S(x,y) = \frac{p(x) \cap p(y) \cap B}{P(x) \cup p(y) \cap B}} \quad \textbf{.....Equation 2.5}$$

$$\boldsymbol{p(x) = \{Pi | x possesses Pi\}} \quad \textbf{.....Equation 2.6}$$

One distance measure that focuses on the similarity of two entities with respect to a property subset b can be created by the following similarity measure (Simon *et al*, 2008)

$$\boldsymbol{Sim(x,y) = \frac{|b(x) \cap b(y)}{b(x) \cup b(y)}; with b(x) = \{Pi \in B | x possesses Pi\}} \quad \textbf{.....Equation 2.7}$$

The distance sim (x, y) between two entities is the larger the less similar they are, or vice versa the distance between two entities is the smaller the more similar they are assigned the same weight. The solution to provide coupling measure will be the normalized measure define as normalized **measure** dist(x, y) = 1 – sim(x, y) (Simon *et al*, 2010).

The model proposed by Simon *et al* (2008) al looks at similarity of the various classes. From the definition of similarity it follows some important observations about classes. According to Simon *et al* (2008) coupling depends a lot on the point of view, it is not an attribute by itself, it exists only between pairs of entities, and it is not defined between two entities with respect to an incompatible total property set.

In concepts from measurement coupling theory, this measuring of distances in classes can be described in the following way:

1. The entities between which the distances are to be measured have to be reduced to the considered properties to get an entity model.
2. The empirical relation system (ERS) consists of pairs of entity-models, between which the distances are to be measured.
3. To reach at least the ordinal scale on the ERS there has to exist at least one relation between entity-model pairs that is connected, antis metrical model, and transitive.

The main advantages of distance-measurement are:

1. Distance with respect to a set of properties is an easy imaginable property between two entities. The classical representation problem that is which numbers are assigned to which entities, is changed to the problem of assigning numbers to entity pairs.
2. Detecting distance is one of the most powerful capabilities of the human perception apparatus.
3. Distance has a geo model equivalent that is the Euclidean distance.

4. There exist many statistical techniques for working with distance values, because these are often produced in empirical science.

Further research on coupling by Zhao and Xu (2009) first consider only direct interaction coupling between two methods in a class. Their definition is then expanded to indirect interaction coupling via transitive method invocations. The weaknesses are considered in the following two scenarios;

1. Component coupling. There are four degrees of component coupling between classes (listed below from strongest to weakest) (Farias,Garcia & Lucena, 2012).

(i) Hidden: Component coupling does not manifest itself in code. For instance, if a class c contains a cascaded method invocation such as a.m1().m2(), the type of the object returned by m2 need not be explicit in the interface or body of class c. It can be found in the interface of the class of the object returned by method m1. That is, in order to detect occurrences of hidden coupling where class c is involved; we also have to look at the interfaces of other classes (Farias,Garcia & Lucena, 2012)..

(ii) Scattered: Component coupling manifests itself in the body of the class only (cases c) and d) in the above definition). Consequently, the body of the class has to be searched in order to detect occurrences of this type of coupling.

(iii)· Specified: Component coupling manifests itself in the interface (cases a and b). It is sufficient to search the interface of the class for occurrences of this type of coupling.

(iv) Nil: No component coupling, this is not true since for object oriented systems coupling between components will always exist.

2. Inheritance coupling. There are three degrees of weaknesses of coupling as regards Zhao and Xu inheritance coupling and they are (Farias,Garcia & Lucena, 2012).;

(i) Inheritance modification: Inheriting class changes at least one inherited method in a manner that violates some predefined "good practice" rules. Eder *et al* (2011) provide examples of such rules as "the signature of an inherited method m may only be changed by replacing the type of a parameter of m, say class d, with a descendent

of class d," "an inherited method must not be deleted from the class interface," and "if a method is overridden, the overriding method must keep the same semantics as the overridden method." The predefined rules applied will, to a certain extent, depend on the used design methodology and programming language. When these rules involve the semantics of methods, they are subjective and not easily measured automatically from a static analysis of the design or source code documents. Modification coupling is the strongest type of inheritance coupling because information inherited from the parent class is modified or deleted in a manner which cannot be justified in the context of inheritance

(ii) Signature modification:The implementation of atleast one inherited method and the signature of the method are changed

(iii)Implementation modification: The implementation of at least one inherited method is changed. This degree of coupling is weaker than the previous type because the signature of the method is not changed.

### 2.2.3 Object Coupling Measures

In a research by Hongyu *et al* (2010) they proposed definition of coupling as the degree of direct and indirect dependencies between parts of object oriented software design. To measure coupling in class diagrams there research proposes the following approaches (Hongyu *et al*, 2010):

1. Number of children approach: -Number of children model defines number of sub-classes subordinated to a class in the class hierarchy. This model measures how many sub-classes are going to inherit the methods of the parent class. Number of children model relates to the notion of scope of properties. If number of children model grows it means reuse increases. On the other hand, as number of children model increases, the amount of testing will also increase because more children in a class indicate more responsibility. So, number of children model represents the effort required to test the class, reuse and maintain.

2. Coupling Between Number of Objects:-A class is coupled with another if the methods of one class use the methods or attributes of the other class. An increase of coupling between number of objects indicates the reusability of a class will decrease. Multiple accesses to the same class are counted as one access. Only method calls and variable references are counted.
3. Number of Dependencies In: -The number of dependencies in model is defined as the number of classes that depend on a given class. This model is proposed to measure the class complexity due to dependency relationships. The greater the number of classes that depend on a given class,the greater the inter-class dependency and therefore the greater the design complexity of such a class. When the dependencies are reduced the class can function more independently. The numbers of dependencies identified are :
4. Number of Dependencies In, Number of Dependencies Out:-The number of dependencies out model is defined as the number of classes on which a given class depends. When the model value is minimum the class can function independently.
5. Number of Dependencies Out, *Number of Association:* - The number of association per class model is defined as the total number of associations a class has with other classes or with itself. When the number of associations are less the coupling between objects are reduced.
6. Direct Dependency: - The direct dependency is direct connection between services. This kind of dependency may exist between services directly when a service calls other services or a service is called by other services. Direct dependency share the local data without any mediator.
7. Indirect Dependency: - The indirect dependency between services may occur at:
   (i) When there is indirect connection or transitive connection between services and
   (ii) When there is a sharing of global data between services.

In a research Darcy and Kemerer (2005) transforms the definition by Wand and Weber, Darcy and Kemerer conclude “any evidence of a method of one object using

methods or instance variables of another object constitutes coupling (Darcy & Kemerer, 2005). As a model for coupling measurement, they define coupling between objects (CBO) as proportional to the number of non-inheritance related couples with other classes. While this idea has been widely appreciated in this literature in principle, some deficiencies have been identified, notably that it does not scale up to higher levels of modularization (Darcy & Kemerer ,2005). Moreover, we may note that coupling is defined as an attribute of pairs of objects, but as a model, it is aggregated to the total number of couples that one class has with other classes, thus implicitly assuming that all basic couples are of equal strength. This does certainly not hold, e.g., when message passing is intermingled with direct access to foreign instance variables. This has generally been identified as the worst type of coupling, also in traditional design. But even if only message passing is considered, several authors have distinguished various forms of coupling with different strengths (Darcy & Kemerer, 2005). For example, sending messages to objects is considered superior to sending messages to one of an object's components, even if the selection of such a component is accomplished by means of an access message.

In a research by Eder *et al* ( 2011) they identify the following types of relationships :

1. Interaction relationships between methods: This type of relationship is caused by message passing.
2. Component relationships: Each object has a unique identifier (the object identity). An object o may reference another object p using the identifier of object p.This introduces a component relationship between the classes of o and p (Eder *et al*, 2011).
3. Inheritance relationships between classes.

From these relationships three dimensions of coupling are derived (Krishnapriya & Ramar,2010 ).

1. Interaction coupling: Two methods are interaction coupled if one method invokes the other, or they communicate via sharing of data. Interaction coupling between method m implemented in class c and method m implemented in class c contributes to interaction coupling between the class's c and m (Eder *et al*, 2011).
2. Component coupling: Two classes C1 and C2 are component coupled, if c¢ is the type of either an attribute of c, or an input or output parameter of a method of c, or a local variable of a method of c, or an input or output parameter of a method invoked within a method of c.
3. Inheritance coupling: Two classes' c and c¢ are inheritance coupled, if one class is an ancestor of the other.

For each dimension of coupling, the authors identify different strengths of coupling (listed below from strongest to weakest) (Eder *et al*, 2011).

1. Interaction coupling: The definition of interaction coupling is most similar to the original definition of coupling. Implementation here means the nonpublic part of the class interface. In C++, for instance, a method m may be declared "friend" of a class c. Method m can then invoke private methods of class c. Access to implementation constitutes a breach of the information hiding principle and is considered the worst type of coupling. There model identifies the following major types of coupling in software systems Eder *et al* (2011).

(i) Common: Methods communicate via unstructured, global, shared data space. The authors cannot give an example for an unstructured global shared data space, because there are no object-oriented languages which support them. Apparently, this type of coupling is only listed in order to be consistent with Myers' categories.

(ii) External: Methods communicate via structured, global, shared data space (e.g., a public attribute of a class). Eder *et al*. find that this is a violation of the locality principle of good software design.

(iii) Control: Methods communicate via parameter passing only, the called method controls the internal logic of the calling method. For instance, the called method may

determine the future execution of the calling method. A change of the implementation of the called method will most likely affect the calling method (change dependencies).

(iv) Stamp: Methods communicate via parameter passing only, the called method does not need all of the data it receives. This constitutes an avoidable dependency between methods. E.g., if the data structure of a parameter of a method is changed, possible effects of this change on the method have to be considered. If the parameter is unused, a change of the parameter's data structure will not have any effects. The effort spent to discover that the change has no effects can be saved by avoiding stamp coupling.

(v) Data: Methods communicate via parameter passing only, the called method needs all the data it receives. This is the best type of coupling (besides no coupling at all), because it minimizes the change dependencies (Eder *et al*, 2011).

2. Component coupling. There are four degrees of component coupling between classes (listed below from strongest to weakest).

   (i) Hidden. Component coupling does not manifest itself in code. For instance, if a class c contains a cascaded method invocation such as a.m1 ().m2 (), the type of the object returned by m2 need not be explicit in the interface or body of class c. It can be found in the interface of the class of the object returned by method m1. That is, in order to detect occurrences of hidden coupling where class c is involved, we also have to look at the interfaces of other classes.

   (ii) Scattered. Component coupling manifests itself in the body of the class only (cases c) and d) in the above definition). Consequently, the body of the class has to be searched in order to detect occurrences of this type of coupling.

   (iii) Specified. Component coupling manifests itself in the interface (cases a) and b)). It is sufficient to search the interface of the class for occurrences of this type of coupling.

   (iv) Nil. No component coupling.

3. Inheritance coupling: There are four degrees of inheritance coupling (listed below from strongest to weakest) (Eder *et al*, 2011).

   i. Modification: Inheriting class changes at least one inherited method in a manner that violates some predefined "good practice" rules. Eder *et al* (2011) provide examples of such rules as "the signature of an inherited method m may only be changed by replacing the type of a parameter of m, say class d, with a descendent of class d," "an inherited method must not be deleted from the class interface," and "if a method is overridden, the overriding method must keep the same semantics as the overridden method." The predefined rules applied will, to a certain extent, depend on the used design methodology and programming language. When these rules involve the semantics of methods, they are subjective and not easily measured automatically from a static analysis of the design or source code documents.

   Modification coupling is the strongest type of inheritance coupling because information inherited from the parent class is modified or deleted in a manner which cannot be justified in the context of inheritance. Two types of modification coupling exist (Eder *et al*, 2011). Signature modification: Not only the implementation of at least one inherited method is changed, but the signature of the method is also changed (Eder *et al*, 2011). Implementation modification: The implementation of at least one inherited method is changed. This degree of coupling is weaker than the previous type because the signature of the method is not changed (Eder *et al*, 2011).

   (ii). Refinement: Inheriting class changes at least one inherited method but the change is made adhering to the predefined "good practice" rules (Eder *et al*, 2011). Refinement coupling is weaker than modification coupling because the inherited information is changed only according to the predefined rules. However, problems can still occur as a result of refinement coupling, e.g. changes to the signature of an inherited method will restrict the use of polymorphism even if the intended semantics of the method are not changed.

Again, like modification coupling, there exist two different types of refinement coupling (Eder *et al*, 2011).

(iii). Signature refinement: Not only the implementation of at least one inherited method is changed, but the signature of the method is also changed (Eder *et al*, 2011). With regards to implementation refinementit was noted thatThe implementation of at least one inherited method is changed. This degree of coupling is weaker than the previous type because the signature of the method is not changed (Eder *et al*, 2011).With regard to extension it was noted that inheriting class changes neither the signature nor the body of any inherited method; only new methods and attributes are added (Eder *et al*, 2011).with nil coupling it was stated that there is no inheritance relationship between two classes.

In a research conducted by Agerfalk, Fitzgerald & Slaughter ( 2009) they revise the Darcy and Kemerer‘s metrics suite. Some of the metrics are adapted or extended, in order to make them applicable to the Aspect soft wares. They introduce the following coupling measures (Agerfalk, Fitzgerald & Slaughter, 2009) :

Weighted Operations in Module (WOM): It counts number of operations in a given module. WOM captures the internal complexity of a module in terms of the number of implemented functions. How much time and effort is required to develop the module is predicted by complexity and number of operations involved in the module. Modules with large numbers of operations are likely to be more application specific, limiting the possibility of reuse.

1. Depth of Inheritance Tree (DIT): Depth of inheritance of a class is its depth in the inheritance tree, if multiple inheritances are involved. It is the length of the longest path from a given module to the class/aspect hierarchy root. The deeper a module is in the hierarchy, the greater the number of operations it is likely to inherit, making it more complex to predict its behavior. Deeper trees constitute

greater design complexity, since more operations and modules are involved. The deeper a particular module is in the hierarchy, the greater the potential reuse of inherited operations ( Cataldo & Nambiar, 2012).

2. Number of Children (NOC): NOC is the number of immediate sub- classes or sub aspects of a given module. The number of children of a module indicates the proportion of modules potentially dependent on properties inherited from the given one. Greater the number of children then greater the reuse due to inheritance. If a module has a large number of children, it may be a case of misuse of sub-classing. If a module has a large number of children, it may require more testing of the operations in that module (Cataldo & Nambiar, 2012).
3. Coupling on Advice Execution (CAE): It is the number of aspects containing advices possibly triggered by the execution of operations in a given module. There is an (implicit) dependence of the operation from the advice. Thus, the given module is coupled with the aspect containing the advice and a change of the latter might impact the former. This kind of coupling is absent in OO systems (Ressia *et al*, 2013).
4. Coupling on Method Call (CMC): It is the number of modules or interfaces declaring methods that are possibly called by a given module. If we use high number of methods from many different modules indicates that the function of the given module cannot be easily isolated from the others. High coupling is associated with a high dependence from the functions in other modules(Lu *et al* ,2012). Examples, constructor calls are counted as a method call; calls from introduced methods are counted as a call from aspect, introduced method calls are counted as an aspect's member calls (Li *et al* ,2013).
5. Coupling on Field Access (CFA): It is the number of modules or interfaces declaring fields that are accessed by a given module. CFA measures the dependences of a given module on other modules, but in terms of accessed fields. In OO systems this metric is usually close to zero, but in AOP, aspects might access class fields to perform their function. Examples, field access from

introduced methods are counted as access from aspect; access to introduced fields is counted as an access to aspect's fields (Kayarvizhy & Kanmani,2013).

6. Response for a Module (RFM): In this, the number methods and advices potentially executed in response to a message received by a given module. The main reason to apply it to AOP software is associated with the implicit responses that are triggered whenever a point cut intercepts an operation of the given module if a large number of methods can be invoked in response to message, the testing & debugging of the module becomes difficult. The larger the number of methods that can be invoked from a class, the greater the complexity of the class (Towne & Herbsleb, 2012).
7. Coupling between Modules (CBM): It is the number of modules or interfaces declaring methods or fields that are possibly called or accessed by a given module. This is a combination of CFA and CMC metrics (Saied *et al*,2015).

In a research by Sherif *et al*, (2011) on coupling measurement, Two design models are considered by authors.

1. Static: can only calculate design time properties of an application.
2. Dynamic: used to calculate actual runtime properties of an application.

From the above two design concepts, two types of coupling is considered by authors;

1. Class level coupling (CLC): Only invocations from one method to another are considered.
2. Object level coupling (OLC): The invocations from one method to another and frequency of invocations at run time is also considered.

The authors also considered that, there is correlation between the number of faults and complexity of system. Therefore, static complexity is used to assess the quality of software. To measure dynamic complexity Model authors used ROOM design charts,

cyclomatic complexity, operation complexity is calculated from ROOM charts (Sherif *et al*, 2011).

Authors explained export and import object coupling with context, description, formula and impact on design quality attributes (Sherif *et al*, 2011).

1. Export Object Coupling (EOC): Measure is a percentage of number of messages sent from object A to object B with respect to total number of messages exchanged during the execution of some scenario (Sherif *et al*, 2011).

2. Import Object Coupling (IOC): Measure is a percentage of the number of messages received by object A and was sent by object B with respect to the total number of messages exchanged during the execution of some scenario (Sherif *et al*, 2011).

### 2.2.4 Dynamic and Static Coupling Measures

The model described by Erik considered following points regarding coupling measurement (Erik, 2011).

1. Dynamic behavior of software can be precisely inferred from run time information.
2. Static coupling measures may sometimes be inadequate when attempting to explain differences in changeability for object oriented design.

The author derived three dimensions of coupling (Erik, 2011).

1. Mapping: object or class
2. Direction: import or export
3. Strength: number of dynamic messages, distinct methods, or distinct classes.

The empirical evaluation of the proposed dynamic coupling measure consists of two parts. The first part is to assess fundamental properties of the measure second part evaluates whether the dynamic coupling measures can explain the change proneness of

class. Erik used the concept of role models for dynamic coupling measurement. The role models are

1. Scenario: a specific sequence of interactions between the objects.
2. Role: abstract representation of the functional responsibility of each object in a given scenario.

According to the authors the Object can have many roles because it may participate in many scenarios. The role model reflects the dynamic coupling between the roles along three orthogonal dimensions: direction, mapping and strength (Erik, 2011).

1. Direction of Coupling (Import and Export coupling): Dynamic import coupling counts the messages sent from a role, whereas dynamic export coupling counts the messages received (Erik, 2011).
2. Mapping: Object-level and Class-level Coupling: Object-level coupling quantifies the extent to which messages are sent and received between the objects in the system. Dynamic, class level coupling quantifies the extent of method dependencies between the classes implementing the methods of the caller object and the receiver object (Erik, 2011).
3. Strength of Coupling: The strength of coupling quantifies the amount of association between the roles. It is of three types (Erik, 2011)..

(i) Number of dynamic messages: Within a run-time session, to count the total number of times each message is sent from one role to another to implement a certain functional scenario.

(ii) Number of distinct method invocations: To count the number of distinct method invocations between two roles.

(iii) Number of distinct classes: To count the number of distinct classes.

Dynamic coupling is compared with static coupling and three important differences are scope ofmeasurement, dead code, polymorphism. In all three measures dynamic coupling is considered more suitable than the static coupling (Erik, 2011).

In a research conducted by Briand *et al* (2004) the authors described many significant dynamic coupling measures and highlights way in which they differ from static measures. The authors collected the measures using UML diagrams and accounted precisely for inheritance, polymorphism and dynamic binding. According to Briand *et al* (2004) classification of dynamic coupling measures.

1. Entity of measurement: The entity of measurement may be a class or an object.
2. Granularity: The granularity can be class level or object level.
3. Scope: The objects and classes are to be accounted for measurement.

The authors captured situations for import and export coupling.

1. Dynamic messages: Total number of distinct messages sent from one object to other objects and vice versa, within the scope considered.
2. Distinct method invocations: Number of distinct methods invoked by each method in each object.
3. Distinct classes: Number of distinct classes that a method in a given object uses.

The authors described much more regarding polymorphism and dynamic binding using the above measures than the others and concluded that coupling is one of the factors which affect change proneness. The authors identified six criteria of dynamic coupling measures (Briand *et al*, 2004).

1. The type of connection: What items (attribute, method or class) constitutes coupling (Briand *et al*, 2004).
2. The locus of impact: To decide whether to count import or export coupling (Briand *et al*, 2004).

3. Granularity of the measure: The level of detail at which information is gathered, i.e. what components are to be measured and how exactly the connections are counted (Briand *et al*, 2004).
4. Stability of server: There are two categories of class stability defined. The first is unstable classes which are subject to modification or development (user defined) and stable classes which are not subject o to change (library) (Briand *et al*, 2004).
5. Direct or indirect coupling: To decide whether to count direct or indirect coupling. For example, if a method $m_1$ invokes a method $m_2$, which in turn invokes a method $m_3$, we can say that m1 indirectly invokes $m_3$. Methods $m_1$ and $m_3$ are indirectly connected (Briand *et al*, 2004).
6. Inheritance: inheritance-based vs. non-inheritance-based coupling, and how to account for polymorphism, and how to assign attributes and methods to classes (Briand *et al*, 2004).

A varied approach for coupling measurement is proposed by Sant'Anna *et al* (2007) .The authors proposed a new model Lack of Coupling in Operations(LCOO) which measures the amount of method/advice pairs that do not access to the same instance variables (Sant'Anna *et al*, 2010). It is an extension of the well-known LCOM (Lack of Coupling in Methods) model developed by Darcy and Kemerer (2005). This model measures the lack of coupling of a component (class and aspect). According to their approach, consider a component C1 with operations (methods and advice) Oi, …, On.

Let {Ij}= set of instance variables used by operation( Sant'Anna *et al*,2007). There are n**such sets {I }, {I }, ….., {I }. Let**

$$|\boldsymbol{P}| = \{(\boldsymbol{I})|\boldsymbol{I} = \emptyset\} \boldsymbol{and} |\boldsymbol{Q}| = \{(\boldsymbol{I})|\boldsymbol{I} = \emptyset\} \boldsymbol{then} \quad \textbf{……..Equation 2.8}$$

$$\boldsymbol{LCOO} = \int |\boldsymbol{P}| - |\boldsymbol{Q}| \boldsymbol{if}\ |\boldsymbol{P}| > |Q| \quad \textbf{…….. Equation 2.9}$$

A high LCOO value, according to Sant' Anna et al, indicates disparateness in the functionality provided by the aspect. The definition of LCOO Model is almost

operational (Sant'Anna *et al*, 2007). It is not stated whether inherited operations and attributes are included or not, and we have to assume that sets {I} only include attributes of j component $C_1$.

Definition of this model is based on AspectJ-like programming languages whose approach to software development do not strictly follow object oriented software design (Sant'Anna *et al*, 2007).

In a similar research Gupta,Singh and Sharma (2015) they propose a coupling metrics using the concept of Strong Functional Coupling (SFC) and Weak Functional Coupling (WFC) and then Gupta,Singh and Sharma (2015) redefined this module coupling metrics. Gupta,Singh and Sharma (2015) commence the dynamic coupling measurement using program execution based approach on the basis of dynamic slicing (dynamic slice is the set of all statements whose execution had some effects on the value of a given variable). They use dynamic slices of outputs to measure module coupling. Their approach addresses the limitations of static coupling metrics by considering dynamic behavior of the programs and designing metrics based on dynamic slices obtained through program execution (Hammad, 2015). They defined SFC as module coupling obtained from common defuse pairs of each type common to the dynamic slices of all the output variables and WFC as module coupling obtained from defuse pairs of each type found in dynamic slices of two or more output variables.

Gethers, Aryani and Poshyvank (2012) proposes a dynamic coupling metrics for GUI programs. According to their research though Graphical User Interfaces (GUIs) make the software easier to use from user's viewpoint however they increase the overall complication of the software since GUI programs unlike conventional software are event based systems. The special characteristics of a GUI program imply that the traditional methods of evaluating complexity statically may not be the suitable ones as static analysis of source code emphasize only on the probability that what may happen when

the program is executing whereas a dynamic analysis attempts to enumerate what actually happened during program execution (Gethers,Aryani & Poshyvank,2012) .

A further research conducted by McConahy *et al* (2012) outline a new technique for collecting dynamic trace information from Java GUI programs and a number of simple runtime metrics are proposed. The exPubMet.Ob metric gives an estimation of level of coupling present in a GUI program and ThepriMet.ob metric shows that simple programs devote a greater proportion of their method access to the internal working of their classes than the GUI program (McConahy *et al* ,2012).The measurement proposed uses two approaches (McConahy *et al* ,2012).

1 exPubMet.Ob: measure of the level of coupling within a program at runtime. = Number of External Public methods called/Total Number of Objects created (Mishra,2013)
2 priMet.ob : measure of the level of cohesiveness within a program. = Number of Private methods called/Total number of objects created (Mishra,2013)

In a research conducted by Ujhazi *et al* (2010),the authors proposed a coupling metrics based on C & K metric suite .These are:

1. Coupling on Advice Execution (CAE) counts aspects containing advices possibly triggered by the execution of operations in a given module.
2. Coupling on Intercepted Module (CIM) counts modules or interfaces explicitly named in the point cuts belonging to a given aspect.
3. Coupling on Method Calls (CMC) counts modules or interfaces declaring methods that are possibly called by a given module.
4. Coupling on Field Access (CFA) counts modules or interfaces declaring fields that are accessed by a given module.
5. Response for a Module (RFM) counts operations potentially executed in response to a message received by a given module, and

6. Crosscutting Degree of an Aspect (CDA) counts modules affected by the point cuts and by the introductions in a given aspect.

In summary to assess and evaluate the various coupling measures this research grouped various coupling measures on the basis of various mechanism of implementation as shown in table 2.1 below. A precise comparison of the models is done on the basis of the mechanism to show there are differences in the manner in which coupling is addressed. One reason for this is the different objectives of the models. A second reason is that some of the issues dealt with by one set of authors are considered to be subjective and too difficult to measure automatically. For example, the stability of an individual class (addressed by Hitz and Montazeri) is not something which can be easily determined unless, say, all problem domain classes are classified as unstable. In this section, we discuss in detail the significant differences between the models and what can be learned from these differences and the mechanisms that constitute coupling.The table is as shown below.

**Table 2.1: Mechanism Classification Criteria**

| SNO | Mechanism | Simon *et al* | Hitz and Montazeri | Bieman and Kangs' | Chae *et al* | Sant'Anna *et al* | Briand *et al* |
|---|---|---|---|---|---|---|---|
| 1 | Methods share data (public attributes) | X | | | | | |
| 2 | Methods references attributes | | X | | | | |
| 3 | Methods invokes other methods | X | X | X | X | X | x |
| 4 | Methods receive pointer to other methods | | | | | | x |
| 5 | Class is type of a class attribute(aggregation) | X | X | X | X | X | X |
| 6 | Class is a type method parameter or return type | X | X | X | X | X | X |
| 7 | Class is a type of method local variable | X | X | | | | |
| 8 | Class is a type parameter of a method invoked from within another method | X | | | | X | |
| 9 | Class is ancestor of another class | X | X | | | | |

## 2.3 Critiques of the Measures

Bieman and Kangs research focuses on two important criteria of coupling measure; the interaction pattern and the special method. The strongest part in this model is to consider the interaction between pattern methods. However, the connectivity factor could generate misleading information leading to poor recognition of the interaction pattern (Voas & Zhang, 2009,). Access methods also cause problems for measures that count pairs of methods that use common instance variables. Because access methods usually access only one instance variable, many pairs of methods that do not reference a common instance variable can be formed using access methods (Voas & Zhang, 2009). Thus, the coupling is artificially decreased. The best Solution to pattern recognition problem is to exclude access methods from analysis. Similarly for constructor which typically reference all instance variables, pattern recognition which artificially increases the coupling of the class, this model is not suitable because it generates many pairs of methods that use an instance variable in common. It is therefore better to exclude constructors from analysis (Voas & Zhang, 2009 ).

The definition stated by Hitz and Montazeri to identify the coupling type does not consider the patterns of the interactions among class members, but just count the number of edges in the connected graph. Such simple counting coupling criteria without taking into account the interaction patterns among class members would lead to a different result than expected when considering the various class patterns in program. Similarly this model doesn't consider the number of sub components and method invocation which has a direct influence on coupling in classes. The montazeri model counts edges and doesn't consider the objects that can be created in software system. It's therefore not an appropriate model for measuring class coupling in object oriented software systems.

Briand *et al* proposes four aspects monotonicity, normalization, symmetry and non negativity. The above stated properties are admissible on a ration scale; hence, there is no contradiction between these properties and the definition of coupling measures on a

ratio scale. However, the given properties are not an indicator that a measure that fulfils them all will be useful; it is likely that a measure, which does not fulfill them all, is ill-defined. However, these properties are not sufficient (Gélinas,Badri & Badri, 2006). The following are the gaps created by the four proposals approach by Briand *et al* model (Hintz and Montezeri, 2011).

1. Normalization: A module can have no coupling or can have maximum coupling. And therefore it's not always applicable all the time that coupling in modules can be measured.

2. Monotonicity for non-coupling components: Adding intra-module relationships will increase the module coupling. Additional internal relationships in modules cannot decrease coupling since they are supposed to be additional evidence to encapsulate module components together.

3. Coupling modules: The coupling of a module obtained by putting together two unrelated modules is not greater that the maximum coupling of the two original modules.

4. Non-negativity: The coupling of a module cannot be negative. Even in the worst case, when none of the module components are related, the coupling of the module is 0.

5. Symmetry: The coupling of a module should not be sensitive to the direction of the relation between modules. If there is a relation between module $m_1$ and $m_2$ then the representation of $m_1 \rightarrow m_2$ is equivalent to $m_2 \rightarrow m_1$.

The Chae et al model considers mostly interaction among the various class components and represents this interaction using graphs (Chae *et al*, 2004). However drawing the graph doesn't take into consideration the various instance variables that should be adopted when developing a system (Hintz & Montezeri, 2011). Similarly the

graph drawn cannot clearly indicate the difference between imported and exported classes, methods (Hintz & Montezeri, 2011).

The Chae *et al* approach considers the disjoint sub reference graphs and ignores the possibility of existence of sub components, sub objects and sub methods (Hintz & Montazeri, 2011). In reality components and objects are not usually disjointed but make reference to each other in well designed software systems as proposed by Chae *et al* (Hintz & Montazeri, 2011)

The model proposed by Simon *et al* looks at similarity of the various classes. From the definition of similarity it follows some important observations about classes. According to Simon *et al* coupling depends a lot on the point of view, it is not an attribute by itself, it exists only between pairs of entities, and it is not defined between two entities with respect to an incompatible total property set.

In concepts from measurement coupling theory, this measuring of distances can be described in the following way:

1. The entities between which the distances are to be measured have to be reduced to the considered properties to get an entity model.
2. The empirical relation system (ERS) consists of pairs of entity-models, between which the distances are to be measured.
3. To reach at least the ordinal scale on the ERS there has to exist at least one relation between entity-model pairs that is connected, anti-symmetric, and transitive.

The main advantages of distance-measurement are:

1. Distance with respect to a set of properties is an easy imaginable property between two entities. The classical representation problem that is which numbers are assigned to which entities, is changed to the problem of assigning numbers to entity pairs.

2. Detecting distance is one of the most powerful capabilities of the human perception apparatus.
3. Distance has a geometric equivalent that is the Euclidean distance.
4. There exist many statistical techniques for working with distance values, because these are often produced in empirical science.

Zhao and Xu first consider only direct interaction coupling between two methods. Their definition is then expanded to indirect interaction coupling via transitive method invocations. The weaknesses are considered in the following two scenarios;

1. Component coupling. There are four degrees of component coupling between classes (listed below from strongest to weakest) (Simon *et al*, 2010).

(i) Hidden: Component coupling does not manifest itself in code. For instance, if a class c contains a cascaded method invocation such as a.$m_1$ ().$m_2$ (), the type of the object returned by $m_2$ need not be explicit in the interface or body of class c. It can be found in the interface of the class of the object returned by method $m_1$. That is, in order to detect occurrences of hidden coupling where class c is involved; we also have to look at the interfaces of other classes.

(ii) Scattered: Component coupling manifests itself in the body of the class only (cases c) and d) in the above definition). Consequently, the body of the class has to be searched in order to detect occurrences of this type of coupling.

(iii) Specified: Component coupling manifests itself in the interface (cases a and b). It is sufficient to search the interface of the class for occurrences of this type of coupling.

(iv) Nil: No component coupling, this is not true since for object oriented systems coupling between components will always exist.

2. Inheritance coupling. There are three degrees of weaknesses of coupling as regards Zhao and Xu inheritance coupling and they are(Simon *et al*, 2010) ;

(i) Inheritance modification: Inheriting class changes at least one inherited method in a manner that violates some predefined "good practice" rules. Eder *et al* (2011) provide examples of such rules as "the signature of an inherited method m may only

be changed by replacing the type of a parameter of m, say class d, with a descendent of class d," "an inherited method must not be deleted from the class interface," and "if a method is overridden, the overriding method must keep the same semantics as the overridden method." The predefined rules applied will, to a certain extent, depend on the used design methodology and programming language. When these rules involve the semantics of methods, they are subjective and not easily measured automatically from a static analysis of the design or source code documents. Modification coupling is the strongest type of inheritance coupling because information inherited from the parent class is modified or deleted in a manner which cannot be justified in the context of inheritance

(ii) Signature modification: Not only the implementation of at least one inherited method is changed, but the signature of the method is also changed.

(iii) Implementation modification: The implementation of at least one inherited method is changed. This degree of coupling is weaker than the previous type because the signature of the method is not changed.

A high LCOO value, according to Sant' Anna et al, indicates disparateness in the functionality provided by the aspect. The definition of LCOO metric is almost operational (Sant'Anna *et al,* 2007). It is not stated whether inherited operations and attributes are included or not, and we have to assume that sets {I} only include attributes of component C1. Definition of this metric is based on Aspect-like programming languages whose approach to software development do not strictly follow object oriented software design (Sant'Anna *et al,* 2010).

### 2.4 Previous Applications of Coupling Measures in Systems

Bartsch and Harrison, (2008) reports the results from a comparative case study of three software development projects where the effect of TDD on program design was measured using object oriented metrics.

The results show that the effect of TDD on program design was not as evident as expected, but the test coverage was significantly superior to iterative test-last development (Bartsch & Harrison, 2008).

Simon and Parmea (2008) express existing definitions of coupling metrics using call graphs. They then compare the results of four different call graph construction algorithms with standard tool implementations of these metrics in an empirical study. Their results show important variations in coupling between standard and call graph-based calculations due to the support of dynamic features.

Hongyu *et al* (2010) primary studies for this review were identified based on a pre-defined search strategy and a multi-step selection process. Based on their research topics, they have identified four main categories of themes: software trends and patterns, evolution process support, evolvability characteristics addressed in OSS evolution, and examining OSS at software architecture level.

Zhoa *et al* (2010) describe about the Object Oriented models that are investigated include cohesion, coupling, and inheritance metrics.Their results, based on Eclipse, indicate that :

1. The confounding contribution of class size on the associations between OO metrics and change-proneness, in general, exists, regardless of whichever size metric is used;
2. The confounding effect of class size generally leads to an overestimate of the associations between OO metrics and change proneness; and
3. For many OO metrics, the confounding effect of class size completely accounts for their associations with change-proneness or results in a change of the direction of the associations.

These results strongly suggest that studies validating object oriented metrics on change-proneness should also consider class size as a confounding variable.

Agrawal *et al*. (2009) suggest an approach to identify vulnerable classes in object oriented design. The method proposed also investigates whether transitive nature of inheritance contributes to propagation of vulnerabilities from one class to another or not. An algorithm for computing Vulnerability Propagation Factor (VPF) was developed, which measures number of vulnerable classes because of the Vulnerability in some classes of an object oriented design (Agrawal *et al*, 2009)

## 2.5 Research Gaps

In the literature review conducted it was noted that every research work had its own component that they were considering for assessment of coupling ,i.e Darcy and Kemerer model define coupling between objects and there measure is only considering objects and which doesn't scale up to modularization ( Darcy & Kemerer, 2005). Eder *et a*l use the definition of coupling provided by Stevens *et al*, the researchers only identify coupling between methods and components that create a program (Eder *et al*, 2011). The model by Hitz and Montazeri only considers coupling between class and objects and therefore provides object level coupling and class level coupling. Using these examples we can easily note that there is no model that considers overall contribution of class, objects, static behaviors and dynamic behaviors in coupling.

This research will address this gap by creating a unified model consisting of class level coupling, object level coupling, dynamic level coupling and static level coupling .The research proposes to collect and analyse data from existing systems to determine the contribution of each component in software systems and generate a model with weights of each factor into the model.

In the reviewof previous research work and publications , it was noted that Hitz and Montazeri approach coupling by defining the state of an object (the value of its attributes at a given moment at run-time), and the state of an object's implementation (class interface and body at a given time in the development cycle) (Arisholm *et al*, 2007).

Consequently a model by Erik Arisholm considers points regarding coupling measurement of dynamic behavior and static behaviours. In a model by Erik Arisholm, Lionel C. Briand, and AudunFøyen ,the authors described many significant dynamic coupling measures and highlights way in which they differ from static measures (Arisholm,Briand & Foyen,2004). The authors collected the measures using UML diagrams and accounted precisely for inheritance, polymorphism and dynamic binding (Vanderfeesten *et al*,2007). From the above examples it can be noted that the authors majorly focused on measure as key contribution without providing an algorithm to implement the measures (Arisholm,Briand & Foyen,2004).

Previous researchers do not provide algorithms to operationalize their research.In this research an algorithm will be presented to operationalize the model developed. Model proposed by previous researchers majorly deal with definition and measures as stated in this section.Most of the models do not have a theortical validation of data.This weakness represents a research gap since there is no way of validating their work. In this research, this gap will be addressed by proposing a model for assessment, perform empirical validation, create a tool to collect data and analyse it, present case studies for assessment and provide results of assessments as validation output

## 2.6 Conclusion

There are different types of coupling among classes, methods and attributes. Methods can be coupled more or less strongly, depending on the type of connections between them the frequency of connections between them. A distinction can be made between import and export coupling.

From the review it can be seen that there exists a variety of decisions to be made during the definition of a coupling measure. It is important that decisions are based on the intended application of the measure if the measure is to be useful. When no decision

for a particular aspect of coupling can be made, all alternative measures should be investigated.

This research proposes to generate a measurement model that will encompass the following aspects of coupling in object oriented software models:

1. Component coupling between the various classes in an object oriented software system
2. Inheritance coupling between the various classes and objects that exist between various software systems
3. Aggregation coupling that exist between various software systems
4. Export and import coupling that exists between the various software classess especially in instances where one components or interface of one software system makes reference to another component of the software system.

# CHAPTER THREE

# 3.0 RESEARCH METHODOLOGY

## 3.1 Introduction

A research process was adopted to carry out his work. Data from software systems were collected within JKUAT and Github.The softwares developed within JKUAT were used earlier in the development of the model while softwares from github were used in the validation of the model. JKUAT was considered appropriate since it would make it easier to get software for modification unlike industries where proprietary rights would pose a great challenge. A questionnaire was developed for use in this section. The software developers were grouped into three to enable development of quality systems. The systems under study were majorly developed in object oriented programing languages.

The software developers and system analyst formed part of the team to be collected data from since they form the largest suppliers, developers and support staff of software.The software developers are also deemed to have greater knowledge in coupling and therefore would help objectively understand the process involved.

The data collected was statistically analyzed using Poisson model .This was to help build the model that was under research in this work.

A Poisson model was selected since were dealing with count data.Hypotheticaly it was expected that a change in one aspect of the program would greatly result in change in a number of components in the system.Poisson model would also make it possible to monitor how fluctuations vary in the software system as a result of contribution of a single occurance of an even in the software.

## 3.2 Research Design

### 3.2.1 Sample Population

The sample populations for this research project are software developers. This group was selected since they are the major producers of software systems for industry and therefore are in a better position to provide correct and accurate data. The software developers were to be engaged in the development of the software systems that would be used in this research.The group was also in a better position to provide and describe the various coupling techniques and challenges they experience during software development. Similarly software developers are engaged in software system maintenance and are therefore in a position to describe the various maintenance challenges they experience during software maintenance process.

The software developers would also double up as system designers and therefore they are in a position to identify the various classes that might exist in software systems, introduce and modify various classes.

A total of 20 softwares were used in this research.A total of 15 softwares were from JKUAT while another five were from Github.Github provided a set of open source softwares used in other countries and therefore would provide best testing environment.

### 3.2.2 Sample Size

To gather enough data that can represent a good number of software's a total of 15 softwares were selected for data collection and analysis and another 5 softwares for validation of the developed model.A total of 45 individual were selected for studies to help collect and analyze data .The sample size was chosen to help get a wider view of data and source for analysis. The 45 individuals formed the group of software developers with varied experience and intrest in software development.The aim was to ensure we build greater synergy required for the development of the software systems.

The diversity of background opens an opportunity to learn and gather new ideas and experiences in software design that are unique to every individual software developer.

The 45 individuals were grouped into three to help create softwares for study in this research. The sample size is considered important since the research is intended to develop a robust model that can support varied users and application developers in the world. The software developers and the softwares were restricted to object oriented software systems to ensure that only relevant and appropriate softwares were selected for the study.

### 3.2.3 Data Collection Methods

The proposed method for data collection in this project is experinmental model with forms to be filled**.** The various classes,objects,methods for software system selected in this study will be changed and corresponding effects in the components of the software system will be recorded.The major data to be collected from the experinment will be count data for use in the development of the model in this study.

### 3.2.4 Data Collection Instrument

An experinment questionnaire form was created as instrument of collecting data.In the form we listed the set of components to be observed if a change occurred in the name and the total count of data of what was affected.The questionaire covered both class components and object components in object oriented software systems selected in this study.The experinment questionnaire was for filling in the counts of the changes in the software system.

### 3.2.5 Data Analysis

A Poisson model was selected since we were dealing with count data as stated in the introduction of this chapter. The poisson parametric table would help provide the β

values required for construction of the model.The procedure to be used to analyze the data collected for each measure will be described in three stages:

(i) Descriptive statistics and outlier analyses,
(ii) Univariate outliers analysis,
(iii) Poisson distribution models

### 3.3 Model and Model Development

The parameter estimates value from Poisson distribution model was used to construct the relationship between the various components of the model. The parameter estimates provided the constant β values for equation generation.

The β were inserted into the Poisson equation model

$$\boldsymbol{Log(Y) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \beta_4 X_4 + \beta_5 X_5 + \beta_6 X_6}$$

to generate the constant of the coefficients for our linear equation.

An algorithm was also created to describe the operation of the model. The algorithm provided a description of how to evaluate the individual components of the model and an overall algorithm for evaluating the entire model was created.

For ease of interaction of the model a tool in java containing the mathematical model was created for ease of interaction within the software. The tool provided a graphical user interface for entry of data and report for final display of data.

### 3.4 Model Validation

For validation purposes the developed model was compared against existing models to assess its level of effectiveness in evaluating coupling in object oriented software systems.To facilitate comparison a baseline model was selected to help classify the

software systems as either highly coupled or lowly coupled. The results obtained were discussed to help explain the theoretical concepts underlying the differences in the values obtained .The validation acted as means to demonstrate the applicability and improvement of the existing model.

### 3.6 Study Hypotheses

The proposed hypotheses that guided the study were the following

a. The current models for measuring software coupling in object oriented system are not effective.
b. Coupling measurement improves programmer productivity in software development.

# CHAPTER FOUR

## 4.0 DATA COLLECTION AND ANALYSIS

### 4.1 Introduction

This chapter covers data collection, analysis and results for this project. The study teams and systems were organized in groups and each component assessed and recorded .The data collected was analyzed using descriptive statistics and Poisson model to establish the various relationships among the components. The software analyzed were from different business application environment to enable the research have wider field application and testing opportunities.

The data was analysed using Poisson model.The model was chosen since the data to be analysed was count data.Poisson model is deemed suitable for count data . The count data was to help generate a Poisson model representing relationships which can be used to get weights for the model equation to be developed in this research.

The following 20 softwares were analysed and studied in this research.The first 15 softwares were used for data collection and analysis during model development while the last five were used for model validation.

1 Text 4 life
2 Fashion mobile application
3 Nanndi County Mobile Payment solution
4 Constituency Fund Management System
5 Agricultural web based Management Information System
6 Kiambu Hospital health records Mnagement Information Systems
7 JKUAT Hospital management System
8 Biometric Fingerprint Based ATM System
9 Nyamira Fleet Management System

10 Nairobi Securities Exchange Online System
11 Electronic Crime Management System
12 Online Help Desk Management System
13 Hostel Management Information System
14 Library Management Information System
15 Transport Management Information System
16 Frontline Sms
17 Open MCT Duka
18 Ushahidi
19 NYE senate Open Legislation
20 Open Duka

### 4.2 Descriptive Analysis

In this section we analyzed descriptive count data for the various components that we studied.

As seen in table 4.1, it was noted that for all the systems tested in this research the minimum number of classes were 6 while the mean number of classes created for the system were 12 amongst all the 15 projects under study in this research. The standard deviation observed about the classes from the data collected was 8.727

**Table 4.1: Classes Created**

| | N | Minimum | Maximum | Mean | Std. Deviation |
|---|---|---|---|---|---|
| How many classes are created within the software developed | 15 | 6 | 14 | 12 | 8.727 |

As regards the class level coupling descriptive statistical testing it was noted that from the various system analyzed and studied in the research as shown in table 4.2 , the mean number of exported classes was 9 with standard deviation being 7.234,the mean number of imported classes were 5 with standard deviation of 2.608,the mean number of attributes affected were 5 with a standard deviation of 2.317,the mean number of new reference created within the software systems were 3 with the standard deviation 1.169,the mean number of new pointers created in the software system was found to be 3 with standard deviation of 1.751,the mean number of aggregations affected in the research were found to be 4 with a standard deviation of 2.898,the mean number of new methods invoked was found to be 6 with a standard deviation of 4.792,the mean number of newly inherited class were found to be 3 with a standard deviation of 1.643.

**Table 4.2: Class Level Coupling Assessment**

| Class level coupling | N | Minimum | Maximum | Mean | Std. Deviation |
|---|---|---|---|---|---|
| No of exported classes | 15 | 2 | 20 | 9 | 7.232 |
| No of imported classes | 15 | 3 | 10 | 5 | 2.608 |
| No of attributes affected | 15 | 1 | 7 | 5 | 2.317 |
| New references created | 15 | 1 | 4 | 3 | 1.169 |
| New pointers created | 15 | 0 | 5 | 3 | 1.751 |
| Aggregation created | 15 | 1 | 8 | 4 | 2.898 |
| No of methods invoked | 15 | 1 | 15 | 6 | 4.792 |
| No of New inherited classes | 15 | 0 | 5 | 3 | 1.643 |

The object level coupling descriptive statistics is as shown in table 4.3From the table in can be noted that a change in object resulted mean number of exported classes being 6 with a standard deviation of 3.082. The mean number of imported classes was 6 with a standard deviation of 2.338 .The mean number of attributes affected was 4 with a standard deviation of 2.066. The mean number of new references created in the software system was 3 with a standard deviation of 2.639. The mean number of new pointers created in the system was 3 with the standard deviation 2.074. The mean number of aggregation created in the software system was found to be 3 with a standard deviation of 1.211. The mean number of methods invoked in the software system was found to be 5 with a standard deviation of 2.639. The mean number of newly inherited class created in the software system was found to be 3 with a standard deviation 1.472

**Table 4.3: Object Level Coupling Assessment**

| Object level coupling | N | Minimum | Maximum | Mean | Std. Deviation |
|---|---|---|---|---|---|
| No of exported classes | 15 | 2 | 10 | 6 | 3.082 |
| No of imported classes | 15 | 3 | 10 | 6 | 2.338 |
| No of attributes affected | 15 | 2 | 8 | 4 | 2.066 |
| New references created | 15 | 1 | 7 | 3 | 2.639 |
| New pointers created | 15 | 0 | 6 | 3 | 2.074 |
| Aggregation created | 15 | 1 | 4 | 3 | 1.211 |
| No of methods invoked | 15 | 0 | 7 | 5 | 2.639 |
| No of New inherited classes | 15 | 2 | 6 | 3 | 1.472 |

As regards static level coupling descriptive assessment as shown in table 4.4 it was noted that mean number of exported classes were 5 with standard deviation of 2.966.The mean number of imported classes was 6 with standard deviation 3.077. The mean number of attributes affected was 5 with standard deviation of 4.167. The mean number new references created was 5 with a standard deviation of 5.19312. The mean for new pointers created was found to be 3 with standard deviation of 2.422. The mean number of aggregation created was 4 with a standard deviation of 1.225. The mean number of methods invoked was found to be 4 with a standard deviation of 3.098. The mean number of newly inherited classes was found to be 2 with a standard deviation of 1.633.

**Table 4.4: Static Level Coupling Assessment**

| Static level coupling | N | Minimum | Maximum | Mean | Std. Deviation |
|---|---|---|---|---|---|
| No of exported classes | 15 | 1 | 10 | 5 | 2.966 |
| No of imported classes | 15 | 1 | 10 | 6 | 3.077 |
| No of attributes affected | 15 | 0 | 11 | 5 | 4.167 |
| New references created | 15 | 1 | 14 | 5 | 3.193 |
| New pointers created | 15 | 0 | 7 | 3 | 2.422 |
| Aggregation created | 15 | 2 | 5 | 4 | 1.225 |
| No of methods invoked | 15 | 1 | 9 | 4 | 3.098 |
| No of New inherited classes | 15 | 1 | 5 | 2 | 1.633 |

With regards to descriptive statistical analysis of data from dynamic level coupling as shown in table 4.5.It was realized that mean number of exported class was 6 with a standard deviation from the study samples at 3.869. The mean number of imported classes stood at 6 with standard deviation of 7.204. The mean of attributes affected stood at 5 with a standard deviation of 3.061. The mean of the new references created was 4 with standard deviation 2.530 ,with regard to new pointers in the system it was noted that mean of 4 and standard deviation 2.639. The number of aggregations created are 4 with standard deviation of 3.688. The mean number of methods invoked was 7 with standard deviation of 6.676.Dynamic level coupling had minimum mean of 2 with a standard deviation of 0.816

**Table 4.5: Dynamic Level Coupling**

| Dynamic level coupling | N | Minimum | Maximum | Mean | Std. Deviation |
|---|---|---|---|---|---|
| No of exported classes | 15 | 1 | 10 | 6 | 3.869 |
| No of imported classes | 15 | 1 | 20 | 6 | 7.204 |
| No of attributes affected | 15 | 2 | 11 | 5 | 3.061 |
| New references created | 15 | 1 | 7 | 4 | 2.530 |
| New pointers created | 15 | 0 | 7 | 4 | 2.639 |
| Aggregation created | 15 | 0 | 8 | 4 | 3.688 |
| No of methods invoked | 15 | 3 | 20 | 7 | 6.676 |
| No of New inherited classes | 15 | 1 | 3 | 2 | .816 |

## 4.2 Generalized Linear Models

In generating the linear model, all the 15 cases for software studied was used in the project. The sample cases data formed input to the Poisson distribution model. The various confidence and parametric tests were done on the data.

Table 4.6 and table 4.7 represents the mean and standard deviation of the continuous variables studied in this research. The dependent variable was the number of classes created in software system and the covariate variable are the various class level coupling (CLC),Object Level Coupling (OLC),Static level Coupling (SLC) and Dynamic level Coupling (DLC)

**Table 4.6: Continuous Variable Information (Mean Factor Consideration)**

| | | N | Minimum | Maximum | Mean |
|---|---|---|---|---|---|
| Dependent Variable | How many classes are created within the software developed | 15 | 6 | 30 | 11.40 |
| Covariate | CLC | 15 | 2.75 | 8.63 | 5.1417 |
| | OLC | 15 | 2.13 | 5.88 | 4.9917 |
| | SLC | 15 | 1.63 | 7.25 | 5.1417 |
| | DLC | 15 | 1.38 | 9.88 | 5.4250 |

**Table 4.7: Continuous Variable Information (Std Deviation Consideration)**

| | | Std. Deviation |
|---|---|---|
| Dependent Variable | How many classes are created within the software developed | 6.266 |
| Covariate | CLC | 1.45610 |
| | OLC | 1.15773 |
| | SLC | 1.55542 |
| | DLC | 1.92133 |

The table 4.8 represents the goodness of fit measure using Pearson chi-square. The Pearson chi square was realized to be 9.590 with a value difference of 1.199themodel: (Intercept), CLC, OLC, SLC, DLC, CLC * OLC, CLC * SLC.Where CLC represented the class level coupling, OLC represented Object level coupling, SLC represented static level coupling and DLC represented dynamic level coupling.The goodness of fit measure was used to ensure that the values we generated was between 0.00-0.1 and that all components assessed in this research would not produce abnormal values.

**Table 4.8: Goodness of Fit Measure**

| | Value | df | Value/df |
|---|---|---|---|
| Deviance | 9.206 | 8 | 1.151 |
| Scaled Deviance | 9.206 | 8 | |
| Pearson Chi-Square | 9.590 | 8 | 1.199 |
| Scaled Pearson Chi-Square | 9.590 | 8 | |
| Log Likelihood[b] | -36.012 | | |
| Akaike's Information Criterion (AIC) | 86.025 | | |
| Finite Sample Corrected AIC (AICC) | 102.025 | | |
| Bayesian Information Criterion (BIC) | 90.981 | | |
| Consistent AIC (CAIC) | 97.981 | | |

Table 4.9 represents the omnibus test that was carried out on the data set for the 15 software systems under investigation. The omnibus test was meant to evaluate likelihood ratio of Chi-Square that Compares the fitted model against the intercept-only model.A significance level of 0.000 was realized from the study with likelihood Ratio chi Square of 29.874

**Table 4.9: Omnibus Test**

| Likelihood Ratio Chi-Square | Df | Sig. |
|---|---|---|
| 29.874 | 6 | .000 |

**Table 4.10: Test of Model Effects**

| **Source** | **Type III** | | |
|---|---|---|---|
| | **Wald Chi-Square** | **Df** | **Sig.** |
| (Intercept) | .630 | 1 | .427 |
| CLC | 3.016 | 1 | .082 |
| OLC | 3.204 | 1 | .073 |
| SLC | 4.369 | 1 | .037 |
| DLC | .098 | 1 | .755 |
| CLC * OLC | 1.341 | 1 | .247 |
| CLC * SLC | 5.295 | 1 | .021 |

The table 4.10 represents the test of the model proposed in this study using the model intercept. In data analysis it was discovered that the intercept of CLC*SLC and SLC had significant value less than 0.05 and therefore they formed the major significant model intercept component. This therefore selects them as the major components for building the proposed model in this research. The software systems under study produced significant effect of high static level coupling and high combination of both static level coupling and class level coupling.

**Table 4.10: Parameter Estimates**

| Parameter | B | Std. Error | 95% Wald Confidence Interval | | Hypothesis Test | |
|---|---|---|---|---|---|---|
| | | | Lower | Upper | Wald Chi-Square | df |
| (Intercept) | 1.090 | 1.3729 | -1.601 | 3.781 | .630 | 1 |
| CLC | .644 | .3710 | -.083 | 1.371 | 3.016 | 1 |
| OLC | -1.112 | .6210 | -2.329 | .106 | 3.204 | 1 |
| SLC | 1.070 | .5121 | .067 | 2.074 | 4.369 | 1 |
| DLC | .045 | .1435 | -.236 | .326 | .098 | 1 |
| CLC * OLC | .156 | .1346 | -.108 | .420 | 1.341 | 1 |
| CLC * SLC | -.235 | .1023 | -.436 | -.035 | 5.295 | 1 |
| (Scale) | 1 | | | | | |

Table 4.11 represents Parameter estimates for all the intercept components in the study together with the β value and various wald chi square values as shown in the table. The β values represent the values for construction for the Poissonmodel to be used to build the predictive model in this research. From the table in can be noted that the overall $\beta_0$=1.090,$\beta_1$=0.64, $\beta_2$=-1.112,$\beta_3$=1.070, $\beta_4$=0.045, $\beta_5$=0.156,$\beta_6$=-0.235

From the table 4.11 the β can therefore be used to form the Poisson formula given by

$$\boldsymbol{Log(Y) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \beta_4 X_4 + \beta_5 X_5 + \beta_6 X_6} \quad \textbf{\textit{.....Equation 4.1}}$$

On substituting the values we have

$$\boldsymbol{Log_e(unified\ coupling\ ) = 1.090 + 0.644\ CLC - 1.112\ OLC + 1.070\ SLC + 0.45\ DLC + 0.156\ CLC * OLC - 0.235\ CLC * SLC}\ldots\ldots\ldots\ldots\ldots\ldots\textit{Equation 4.2}$$

Eliminating the logs to generate the natural logs represented by *e* we have the overall class unified formula represented as

$$\boldsymbol{unified\ coupling} = (\boldsymbol{e})^{\boldsymbol{1.090}} \times (\boldsymbol{e})^{\boldsymbol{0.644}}\boldsymbol{CLC} \times (\boldsymbol{e})^{\boldsymbol{-1.112}}\boldsymbol{OLC} \times (\boldsymbol{e})^{\boldsymbol{1.070}}\boldsymbol{SLC} \times (\boldsymbol{e})^{\boldsymbol{0.45}}\boldsymbol{DLC} \times (\boldsymbol{e})^{\boldsymbol{0.156}}\boldsymbol{CLC * OLC} \times (\boldsymbol{e})^{\boldsymbol{-0.235}}\boldsymbol{CLC * SLC}\ldots\ldots\ldots\ldots\ldots\textit{Equation 4.3}$$

**4.3 Descriptive Analysis**

### 4.3.1 Class CouplingAnalysis

From table 4.2 it was noted that a change in a class affects both import coupling and export coupling with significant values. It is important to note that even though an OO module typically exports nothing, you might choose to export a named constructor or management routine. This routine typically acts a bit like a class method but is meant to be called as a normal routine.

Classes form a hierarchy of parent classes and children classes .A Child is a subclass of, derived class of, inherits from, extends the parent, super class and therefore much importing will have to occur within the various classes in the computer program. Similarly it is important to note that Class variables hold state that is shared in common among all objects of class and Class methods act on attributes of the whole class through class variables therefore Class methods can't access instance variables and they Will only use instance variables and methods here in within the current class they are operating in.

It is also noted that with regards to method invocation a low value was recorded. The underlying principle is that when call is made to the generic method function three things are done:

1. The list of available methods is obtained

2. The methods are arranged in order from most specific to least specific

3. the most specific method is called

The determination of which method is most specific is made based on the classes of the arguments to the methods (and on the class structure that they induce – ie. who do they inherit from). Method of super class may be overridden in subclass. When a method of super class is applied to object of subclass, methods of subclass are used in computing result, instead of methods of super class. For example instance method of subclass may hide (or shadow) method of super class (Wheeldon & Counsell, 2003).

For measuring the class coupling, we only use those public methods which are neither constructor nor destructor for a class. Constructor and methods only serve the purpose of initialization of class attributes and they are not supposed to perform any kind of functionality for consumer environment (Ward & Daniel, 2006 ). Similarly to constructor methods and operator over loader methods will also be ignored because such methods read or write to all attributes of class and consequently act like an initializing method. Inclusion of operator overloading may mislead value of class coupling.

Destructor methods are used when an object is no longer needed it has to be deleted. Whenever an object is deleted its destructor member function is called. Destructors in OOP are normally used when overloading or inheritance not allowed. In using destructors the access modifiers or parameters not to be specified consequently the order of call to destructor in a derived class is from the most derived to the least derived

(Kanellopoulus *et al*,2006). Destructor methods are not only called during object destruction, but also when the object instance is no longer eligible for access

A class that is composed of related methods will result in coupling as its methods will use the same set of attributes in their implementation (Horgan & Martha, 1990). A class that has single public method will turn out be the most coupled, because all of its attributes will be used by the one public method. Bieman and Kang have also called such class as the most coupled class (Bieman & Kang,1999). Following the description some of the possible approaches for finding functionality groups of public methods are:

1. Methods working on a particular member attribute of class may form group of related functionality.
2. Methods using a particulars external resource such as file, database table and memory may form a group of related functionality.
3. Methods using same private, protected and public method may also form group of related functionality.

From table 4.2 it was also noted that the minimum number of inherited classes stood at 0. Most of the time generalizations are used among the classes and interfaces to show inheritance relationship. Generalization is called is- kind-of relationship. This type of relationship exists among the general kind of classes and their more specific kind of classes. In this way, the relationship links generalized class with specialized class. By a specialized class we mean derived class which is also referred as subclass or child class. And, by generalized class we mean base class or parent of the derived class. During OO modeling, the generalized relationships are established when one class is found as a more specific kind of other class.

Notion of the generalization is also referred as inheritance.Dirk (2011) in their research has reported the significance of inheritance or generalization relationship and he has emphasized that OO metrics like cohesion, coupling and size should be studied while considering the effects produced by the inheritance (Dirk *et al.* 2011). In the UML,

generalization relationship can also be created among the packages. This is more applicable during the design of the software system.

In OO modeling there are three types important relationships among the classes, namely dependency, inheritance and association relationships. Dependency represents the using relationships among the classes; inheritance relationship connects generalized classes to their specialized classes; and association relationship shows the structural relationship among the objects (Kalogerakis *et al*, 2006). This therefore results in stronger bonds between various coupled classes which provides higher probability of the values for export and import coupling as observed in this research work as shown in Table 4.2

### 4.3.2 Object Level Coupling Analysis

From Table 4.3 a low standard deviation of less than 4 was recorded for all the various attributes tested in this research. The possible underlying principle that could be major contributing factor is that Coupling between Objects is a measurement which can be interpreted to show the reusability of a component and its proneness to change in the future. This proneness to change is caused through its extensive coupling throughout the system. If one object is modified where the coupled object relied on the preexisting behavior previously, then there is a subtle defect that is potentially introduced. Unnecessary object coupling therefore needlessly decreases the reusability of the coupled objects, consequently unnecessary object coupling also increases the chances of system corruption when changes are made to one or more of the coupled objects (Lanza & Marinescu ,2007).

As the degree of coupling increases so does the complexity of the class. This results with the module becoming increasingly dependent on external classes to implement its functionality and is bound to reflect any changes the external classes may undergo in future maintenance (Lanza & Marinescu ,2007).

The coupling between two objects is also viewed in object oriented programming as the manner and degree of interdependence between objects. The coupling between two objects can be determined using a classical (module-based) taxonomy, regardless of the class hierarchy (MacCormak,Rusnak & Baldwin,2006). Objects correspond to classical modules, and the classical definition of coupling modules applies. For example, suppose that a data element (attribute) of an object is defined to be public. This can induce content coupling, because one object may potentially modify a data element of another object (Marcus,Poshyvank & Ferenc,2008). Similarly, if an object (or a characteristic of an object) is declared to be a friend of another object, then this again can induce content coupling. Similarly in most cases in an object that is not hierarchically related to another object has direct access to the underlying implementation of the other object via mechanisms such as public or open access (Losavio,Matteo & Morantes,2009). However, because inheritance is not involved, these are simply new names for classical categories of coupling between two objects. In OOP systems polymorphism allows the implementation of a given operation to be dependent on the object that contains the operation.

For hierarchically related classes and objects, inheritance is considered because it captures a form of abstraction, which is always called super-abstraction, which complements data abstraction (Lanza & Marinescu,2007). Inheritance can express relations among behaviors such as classification, specialization, generalization, approximation, and evolution (Prikladnicki & Audy,2010). Inheritance classifies classes in much the way classes classify values. The ability to classify classes provides greater classification power and conceptual modeling power in designing and developing object oriented software systems. Classification of classes may be referred to as second-order classification. Inheritance provides second order sharing, management, and manipulation of behavior that complements class and object actions.

In structured design, there were few semantic guidelines to decompose a system into smaller subsystem. Consequently, syntactic aspects like size, coupling etc. played a

major role. In contrast, in the object-oriented paradigm, the main criterion for systems decomposition is the mapping of objects of the problem domain into classes or subsystems in the analysis and design model, thus reducing the relative importance of syntactic criteria (Gelina,Badri & Badri,2006). Object-oriented analysis and design strive to incorporate data and related functionality into objects. This strategy in itself certainly reduces coupling between objects. Therefore, explicitly controlling coupling does not seem to be as important as in structured (especially top-down) design.

In many cases, data or operations cannot be unambiguously assigned to one or another class on the grounds of semantic aspects, thus designers do need some kind of additional criteria for such assignments (Porshyvanyk & Marcus,2006). Although introduction of classes as a powerful means for data abstraction reduces the data flow between abstraction units and therefore reduces also total coupling within a system, the number of variants of interdependency rises in comparison to conventional systems. This can be attributed to (Shen,Zhang & Zhao,2008):

1. The variety of mechanisms (inheritance, delegation, using- and has-relationships etc.) a
2. The diversity of modules (classes and objects as well as functions and procedures in hybrid systems). The different mechanisms can sometimes also be employed interchangeably, e.g., inheritance can sometimes be simulated by delegation, or using-relationships can sometimes be replaced by has-relationships etc. Each of these variants exhibits different impacts on quality attributes which must be investigated and measured.
3. The principles of encapsulation and data abstraction, although fundamental to object orientation, may be violated to different extents via the underlying programming language . This leads to different strength of de-facto coupling which should be taken into account

### 4.3.3 Static Level Coupling Analysis

With regards to static coupling higher values of 10 was reported for imported classes, 11for attributes affected, 14 for exported classes and 14 for references created respectively. This is reportedly contributed by the concept of inheritance. Inheritance is one of the initial features of object oriented programming, and through inheritance a derived class receives the attributes and methods of the base class. The relationship between derived and base class is referred as "is-a/is-a-kind- of" (Tsang,Clarke & Banaissad,2004). Inheritance feature creates a class hierarchy. Nowadays interfaces are heavily used in all disciplines especially in object oriented programming. With interface construct, object oriented programming features a good concept with high potential code reusability (Tsang,Clarke & Banaissad,2004). Interfaces are used to organize code and provide a solid boundary between the different levels of abstraction .It is good to use interfaces in large type of applications because interfaces make the software/program easier to extend, modify and integrate new features. An interface is a prototype for class. Interfaces in object oriented programming just contain names and signatures of methods and attributes, but no method implementations (Tsang,Clarke & Banaissad,2004). Interfaces are implemented by classes. The inheritance hierarchy of interfaces is independent than that of class inheritance tree. Therefore object oriented languages give higher potential to produce reusable code than abstract classes. Interfaces are prototype for a class. Interfaces can be used like classes in declarations and signatures

The table 4.10 represents the test of the model proposed in this study using the model intercept. In the research process it was discovered that the intercept of CLC*SLC and SLC had significant value less than 0.05 and therefore they formed the major significant model intercept component in this research. This therefore selects them as the major components for building the proposed model in this research. The software systems under study produced significant effect of high static level coupling and high combination of both static level coupling and class level coupling.

# CHAPTER FIVE

## 5.0 UNIFIED CLASS COUPLING MODEL AND FRAMEWORK

### 5.1 Introduction

In this chapter a presentation and discussion of the various model generated in the research,an algorithm and model is conducted. The major things produced in this research are the coupling model for assessment in objects oriented programming, a model representation of a model, an algorithms for the model and a tool demonstrating the concepts, evaluation and assessment criteria of the model.

### 5.2 The Proposed Advanced Unified Class Coupling Model

A unified class coupling model was generated based on the values from table 4.11 .The equation of the model generated for unified class coupling is given by $\boldsymbol{unified\ coupling} = (\boldsymbol{e})^{\mathbf{1.090}} \times (\boldsymbol{e})^{\mathbf{0.644}}\boldsymbol{CLC} \times (\boldsymbol{e})^{\mathbf{-1.112}}\boldsymbol{OLC} \times (\boldsymbol{e})^{\mathbf{1.070}}\boldsymbol{SLC} \times (\boldsymbol{e})^{\mathbf{0.45}}\boldsymbol{DLC} \times (\boldsymbol{e})^{\mathbf{0.156}}\boldsymbol{CLC} * \boldsymbol{OLC} \times (\boldsymbol{e})^{\mathbf{-0.235}}\boldsymbol{CLC} * \boldsymbol{SLC}$ ***….Equation 4.3***.

This formula was generated to assess levels of coupling in software systems. The formula involved determining the exponential of class level coupling, object level coupling, static level coupling, dynamic level coupling and combination of CLC and OLC, CLC and SLC. The formula determines their combined effects in coupling in software systems.The various beta exponential values were generated as a result figures obtained from the parametric estimates table.

### 5.3 Proposed Unified Class Coupling Algorithm

To generate the algorithm for proposed unfiedcoupling,It is important to produce an algorithm that describes how to evaluate each and every component of the model.This will provide the much needed insight on how each individual component value was

arrived at.It is also important to note that every component of the model is evaluated independently from each other and so the algorithms will address individual contributions.

### 5.3.1 Class Level Coupling Algorithm

In this research a class that is composed of related methods will result in coupling as its methods will use the same set of attributes in their implementation. With this in mind a class that has single public method will be considered as highly coupled, because all of its attributes will be used by the one public method. This therefore implies that:-.

two classes are related by aggregation relationship, i.e., one class has the other class as type of one of its attribute;

1. one class inherits another class or one class implements an interface;
2. one class has got a method that is invoking method of another class;
3. one class has a method referencing an attribute of another class;
4. one class has got a method having parameter of the type of another class;
5. one class has a method containing a local variable of the type of another class;
6. one class has a method invoking a method having a parameter of the type of another class.
7. Methods working on a particular member attribute of class may form group of related functionality.
8. Methods using a particulars external resource such as file, database table and memory may form a group of related functionality.
9. Methods using same private, protected and public method may also form group of related functionality. The algorithm to be adopted will have components as follows

1. $C_i$ : is an instance of a class
2. $C_j$ : is the set of classes invoking the methods or inheriting methods of other classes
3. $M_i$: the number of methods invoked
4. $M_h$:the number of methods inherited
5. Ma:the number of methods referencing attributes of another class
6. Mp:the number of methods having parameter of the type of another class
7. n; the total number of methods
8. c:total number of classes under study
9. {}:representation of set of all elements considered in the algorithm

$$\boldsymbol{CLC_{(i,j)} = \sum_{i=1}^{i=n}\left(\frac{m_h+m_a+m_p+m_j}{n*c}\right)} \ldots \textbf{\textit{Equation 5.1}}$$

Therefore the algorithm for measuring CLC will be as follows

1. Determine the set of classes to study and evaluate
2. determine if one class inherits the attributes of another class or one class implements an interface of another class
3. D1: one class inherits another class or one class implements an interface;
4. *IF* D1== TRUE then
5. Class a methods invokes methods of another class b
6. mj=the number of methods invoked
7. Class a reference the attributes of class b
8. ma=the number of attributes referenced
9. mp=the number of parameters of type of class a in class b
10. mh=number of methods inherited

add all the values numerated for all classes and divide by the total number of classes under study in the research using Equation 5.1

**Figure 5.1 :Class level coupling Algorithm**

### 5.3.2 Object Level Coupling Algorithm

$OLC_X$ is a measure of mutual coupling between two specific objects. Mutual coupling between objects is important for identifying possible sources for exporting errors, identifying tightly coupled objects, and testing interactions between object

An object accessing another object will be called client while the object accessed will be referred to as server; the corresponding classes will be called client class and server class, respectively. This definition also applies when an object accesses parts inherited from its own super class (the server class in this case).

By access to the interface of a class we refer to sending messages according to its method protocol only; in this way, instance variables can never be directly read or modified. On the other hand, the term access to the implementation describes the situation when an instance variable is immediately accessed, either via a direct reference (i.e., by name) or indirectly via a method returning a reference to that variable.

With regard to object coupling the following models are proposed to be used;-

$o_i$ : is an instance of a class (an object)

$O$ : is the set of objects collaborating during the execution of a specific scenario

M:represents count of elements collaborating

$z$: The number of elements in set $Z$

$\{\}$ : A representation of a set of elements

Then we can say that:*OLC of O* is the count of the number of messages sent from $o_i$ to $o_j$ during the execution of a specific scenario $x$.

$$OLC_x(O_iO_j) = \sum_{Z=1}^{Z=n}\left(\frac{M}{Z}\right) \text{ ……...Equation 5.2}$$

Therefore the algorithm for measuring OLC will be as shown below

1. Select an instance of class $o_j$
2. select the set of objects O that are executing at that particular instance of the class $o_j$
3. Create a set {} to include a set of all elements Z identified in the objects under study in the research
4. D1:-Make a decision for all set that are collaborating and those not collaborating,
5. IF :-element collaborating
6. yes then
7. count the element ∑
8. add to set M
9. if No discard the element out of the set.
10. Repeat for all set of the elements in the set Z created that are interacting with the object.
11. Add all the set of collaborating elements and then divide by the total number of elements in the set to get mean coupling between the objects as shown in equation 5.2

**Figure 5.2 :Object level Coupling Algorithm**

### 5.3.3 Dynamic Coupling Algorithm

Dynamic Coupling mainly occurs between packages and it is the manner and degree of interdependence between them. Packages may consist of classes and sub-packages as their elements. Further, these sub-packages may contain classes and sub-packages as their elements. This leads to a hierarchical structure of packages in a software system. Theoretically, every package is a stand-alone unit, but in reality packages depend on

many other packages as either they require services from other packages or they provide services to other packages. Thus, coupling between packages cannot be completely avoided but it can only be controlled. The coupling between packages is an important factor that affects the quality or other external attributes of software, e.g., reliability, maintainability, reusability, fault-proneness etc (Mitchell, 2005). This is determined by the number of classes under assessment in the program under review.

Proposed Algorithm components:

To effectively create algorithms, the following ratio components proposals are made

1. Dynamic Afferent Coupling ($D_{Ca}$): It defines the ratio of number of classes accessing the methods of a class at runtime to the total number of classes. Then we can say that

   $$\boldsymbol{D_{ca} = \frac{C_{am}}{C_t}} \quad \textbf{\textit{..... Equation 5.3}}$$

Where $C_{am}$ is the number of classes accessing methods and $C_t$ is total number of classes

2. Dynamic Key Class ($D_{KC}$): It defines the ratio of sum of calls sent out from the class and calls received by the class at runtime turned on the total number of static calls sent and received by all the classes.

   $$\mathbf{D_{kc} = \frac{C_{sent}}{ST_{calls}}} \quad \textbf{\textit{.....Equation 5.4}}$$

Where ST is total static calls in the class and $C_{sent}$is count of calls sent and received during runtime

3. Percentage Active Classes ($P_{AC}$): It defines the ratio of number of classes sending or receiving at least one method calls from/to another class at runtime to the total number of classes.

   $$\boldsymbol{P_{ac} = \frac{SRM}{TTC}} \quad \textbf{\textit{.....Equation 5.5}}$$

Where SRM is the sum of receiving and sending classess and TTC total number of classes in the program

Representing the three compounds as contribution to the overall DCL we can say that

$$\boldsymbol{DLC = (D_{ca} + D_{kc} + P_{ac}) \quad /n \ldots\ldots\ldots\ldots Equation\ 5.6}$$

Where n is the total number of classes under study in the research

The algorithm to be used is as follows

1. Let TC represent the total number of classes to be studied and assessed
2. determine the various number of methods in the class selected
3. count the number of classes accessing methods at run time
4. divide the count in S1 to TC determined in this process
5. allocate the ratio value to $D_{ca}$
6. Determine the static calls received and sent by the class
7. determine the total number of calls sent and received from the class and add the two sets
8. divide the values in S6 and S5 and label it as $D_{kc}$
9. determine the sum of classes sending or receiving at least one method calls from/to another class at runtime
10. divide value in S8 and TC and label it as $P_{ac}$
11. evaluate DCL as sum of $D_{ca} + D_{kc} + P_{ac}$ divided by TC

**Figure 5.3 Dynamic Coupling Algorithm**

### 5.3.4 Static Level Coupling Algorithm

In this static level of coupling a proposition is made that coupled interactions only happen within the modules of OOP program but not everywhere in the modules. The ration of static level coupling is therefore

$$\mathbf{SLC} = \sum_{\mathbf{c=i}}^{\mathbf{c=n}} \left(\frac{\mathbf{I_m}}{\mathbf{T_m}}\right)/\mathbf{TTC} \text{ ……..} \textit{Equation 5.7}$$

Where $I_{m\ is}$ is the count of all interacting modules in the class under study

$T_m$ is the count of total modules existing within the class under study.

TTC is total number of classes to be studied.

The simplistic model algorithm for a static level coupling would be as follows

1. determine the set of classes to be studied in the research or assessment and assign the values to TTC
2. select a particular class and evaluate the total number of modules within the class and assigning it to Tm
3. Count the number of all interacting modules in the class and assign in Im
4. Divide the number of $I_m$ to Tm
5. Repeat this for all n classes under review
6. Evaluate the static level coupling by adding the values in S4 and dividing by TTC

**Figure 5.4: Static level Coupling Algorithm**

### 5.3.5 Unified Class Coupling Algorithm

From the above calculations and algorithms, it can be noted that arriving at the proposed unified formula requires first dealing with the individual components that make up the

formula. The values cannot be evaluated determine overall unification level in the software system until we establish the individual component value. Similar each component has an algorithm applicable to it and the formula for determining the overall weights and contribution.

In the research the unified algorithm will be as follows.

1. Select a software S to be studied in the research.
2. Determine the total number of classes, objects, methods, modules and attributes in software/program S to be studied
3. Apply equation 5.1 to data collected to evaluate the value of Class Level Coupling
4. Apply equation 5.2 to the data collected to evaluate the value of Object Level Coupling in software S under study
5. Apply equation 5.6 to the data collected to evaluate the value of Dynamic Level coupling in software S under study
6. Apply equation 5.7 to the data collected to evaluate the value of Static level coupling in software S under study.
7. Apply the unified coupling formula generated in equation 4.3 to evaluate the unified class coupling in the software S under study.
8. D1 If $0 \leq U_{cc} \leq 0.0499$ then it implies loose coupling and therefore low error progression and maintenance effort required in future for maintenance and support of the system.

   Else
9. D2 if $0.05 \leq U_{cc} \leq 0.1$ then it implies high coupling in the OOP program and therefore there is a chance of high error propagation and high maintenance effort required in future for the program.

**Figure 5.5:Unified Class Coupling Algorithm**

### 5.4 Proposed Unified Class Coupling Framework

The proposed unified framework uses the various values for evaluation of coupling levels in the software system under study. The values used for calculation are as stated in the coupling algorithms. The unified class coupling evaluator engine applies the Ucc formula to evaluate the coupling levels in different software system. Once the Ucc value has been determined it is used to classify the coupling level in software systems as either high coupling or low coupling. The evaluation and operations of theframework are as shown in the diagram below.

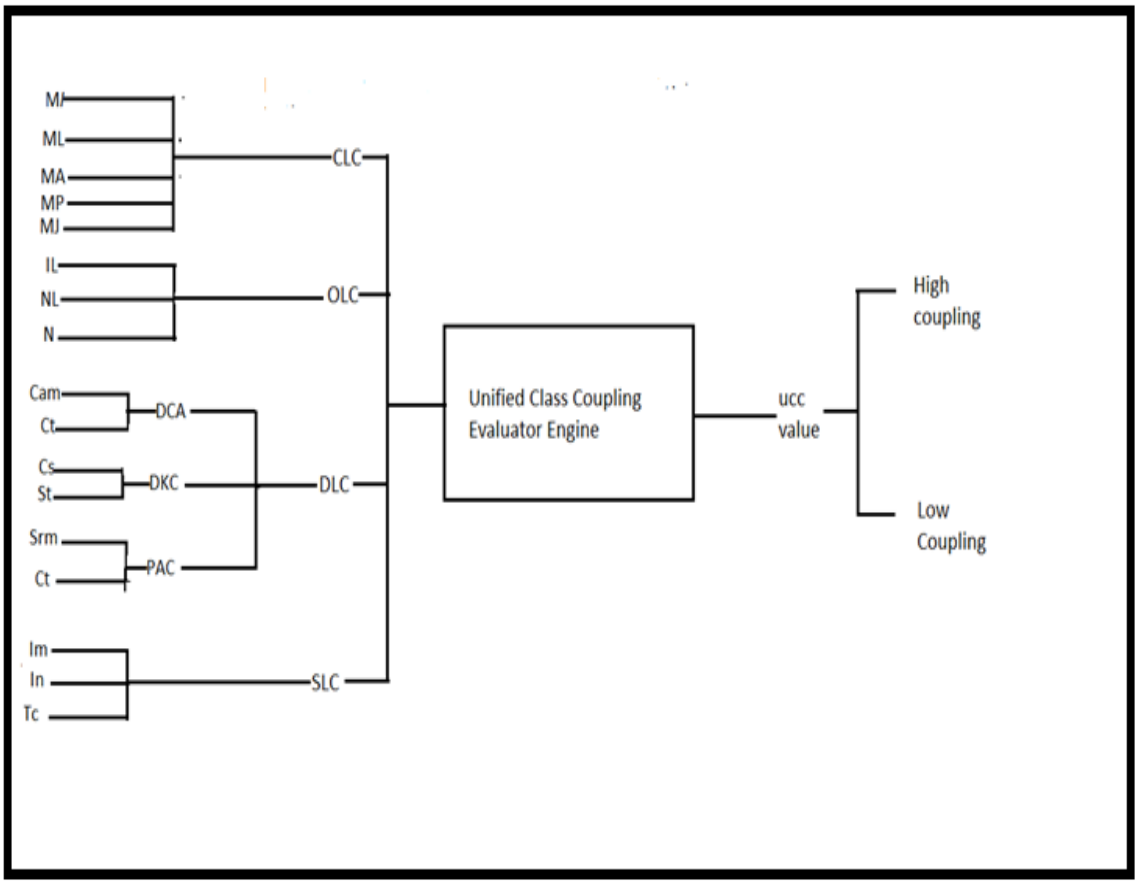



**Figure 5.6:Proposed Unified Class Framework**

### 5.4.1 The Unified Class Coupling Evaluator Engine

The unified class coupling evaluator engine is the major assessment model generated in this research. The model engine receives all the values and parameters entered into it. It then evaluates the individual CLC,SLC,OLC and DLC values and then applies the weighted contribution of each modeler in determining the overall unification coupling value in the software system.The model evaluation function is what was used to build the engine to evaluate coupling values.

During the evaluation, the determinant function for low coupling is any value below or equal to 0.049 while any value above or equal to 0.05 is an indicator of high coupling.The UCC evaluator engine uses these values to classify coupling between the various software tested in this model.

### 5.4.2 Unified Class Coupling Observation

The unified class coupling implements content coupling in its evaluation by counting the number of messages and data sent between different modules. This is taken care of by determining the static calls received and sent by the class and determine the total number of calls sent and received from the class and add the two sets. The values determined here are used in the calculation of Dynamic key Class coupling ($D_{kc}$). $D_{kc}$ value evaluated at this level is a contributor to the overall dynamic coupling level used in the Ucc formular.

The unified class coupling implements and common coupling by determining the various modules that capture and share data. It also considers static modules and dynamic modules in computer programs. This is taken care of during evaluation of static coupling by determining set of modules existing within a class in a study and counting or selecting the set of all interacting modules in those classes. The interacting modules

are considered to be sharing data which is a key feature of common coupling measurement.

The unified class coupling implements data coupling in assessment of application by considering instances where modules share data through parameters. This is implemented for instance in the application by object level coupling where in determining object level coupling ,an object accessing another object will be called client while the object accessed will be referred to as server; the corresponding classes will be called client class and server class, respectively. This definition also applies when an object accesses parts inherited from its own super class (the server class in this case). By access to the interface of a class we refer to sending messages according to its method protocol only; in this way, instance variables can never be directly read or modified. On the other hand, the term access to the implementation describes the situation when an instance variable is immediately accessed, either via a direct reference (i.e., by name) or indirectly via a method returning a reference to that variable.

The unified class coupling implements message coupling which is considered as the loosest type of coupling this is satisfied by SLC that determines the set of interacting modules which communicate via parameters or message passing. The SLC considers set of interacting modules and non-interacting modules in OOP programs.

### 5.5 Unified Class Coupling Evaluation Tool

To advance on the unification and support evaluation of coupling level in software systems, a unified class coupling tool was developed in java to allow us automate the evaluation of coupling in software systems.The tool developed had the following

#### 5.5.1 CLC Calculator

The interface below allows a user to enter the values for the various parameters needed to evaluate class level coupling in the software system.

Once the values has been entered the tool automatically calculates CLC and stores it within its data engine. The user clicks next to move to the next item for evaluation.

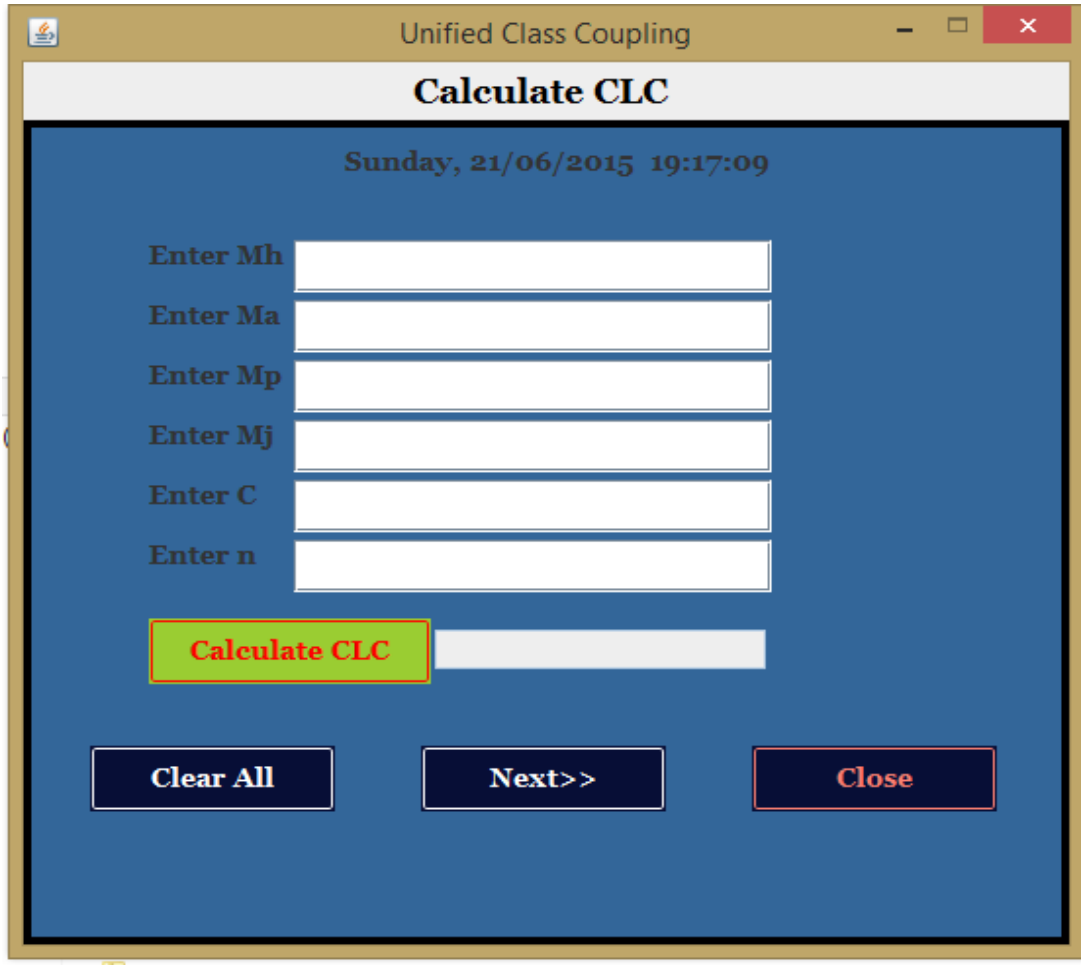



**Figure 5.7: CLC Calculator Interface**

### 5.5.2 OLC Calculator

The interface below indicates the tool section concerned with evaluation of object level coupling. In the calculation the user is only expected to key in the value of M and Z as the number of classes N is automatically loaded from the CLC interface.

Once the OLC value has been calculated and displayed the user clicks on Next to load the form for another component evaluation.

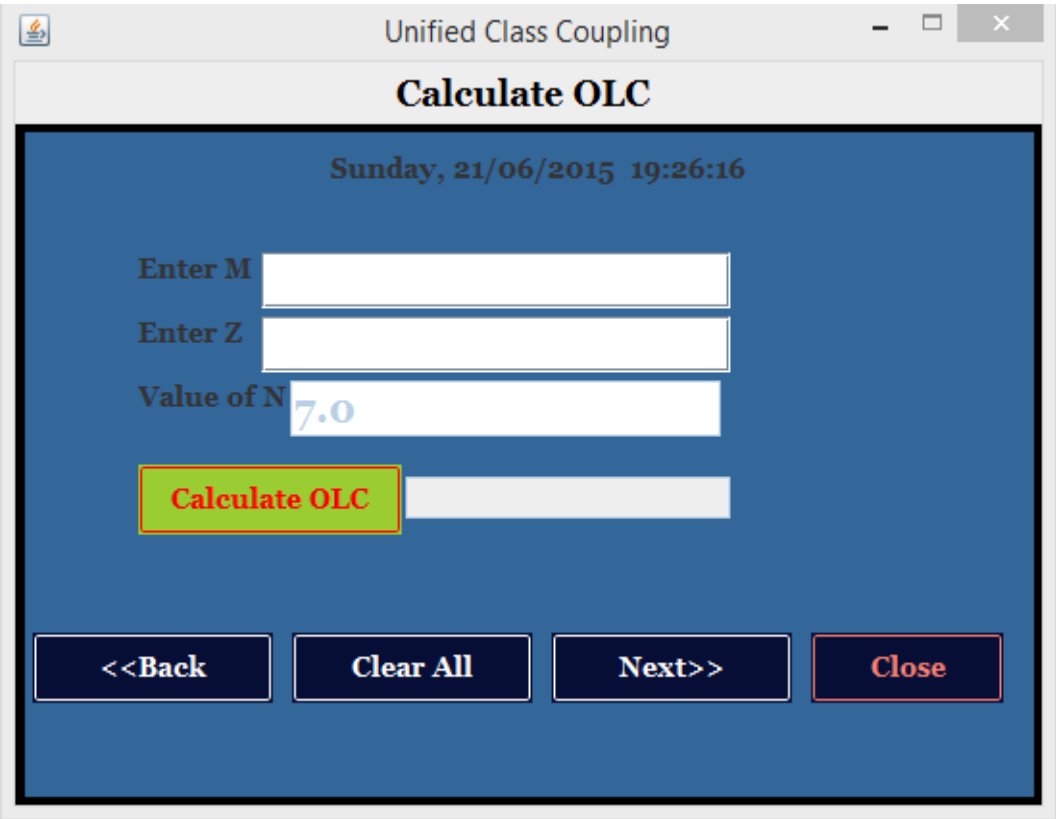



**Figure 5.8: OLC Calculator Interface**

### 5.5.3 DLC Calculator

The interface below is used for calculating Dynamic Level Coupling. The interface provides the capability to calculate and evaluate the individual contributing values before calculating the overall DLC value for the software system that is under assessment and study.

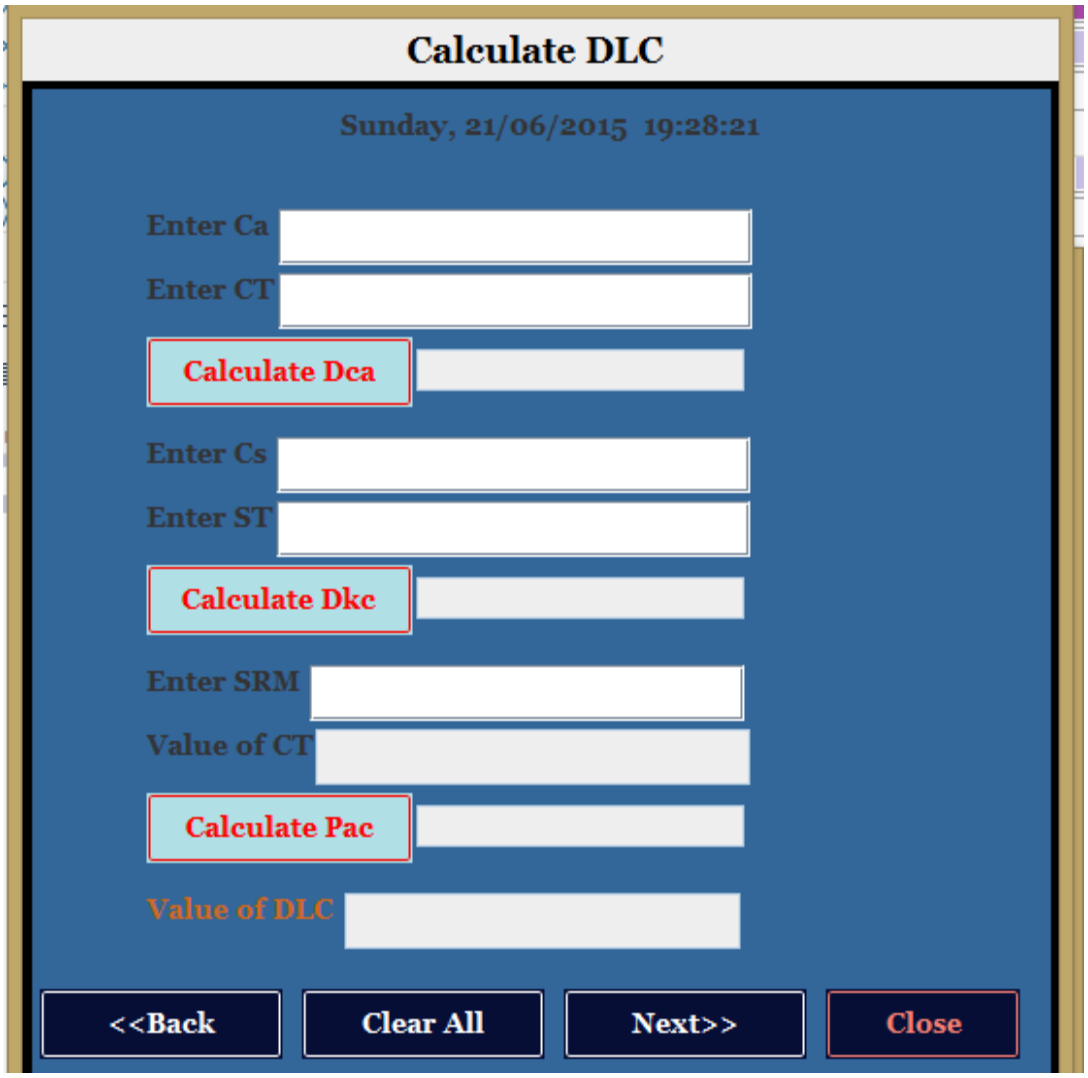



**Figure 5.9: DLC Calculator Interface**

### 5.5.4 SLC Calculator

The interface below is used to display calculated static level coupling for the software system under study. The values to be entered are the values for interacting modules and total modules. The total classes are picked automatically from the software engine for use in evaluation in the software system. Once the values are calculated the user can straight way click on next so that summary calculation are

displayed together with the overall unified class coupling values as indicated in the next interface below.

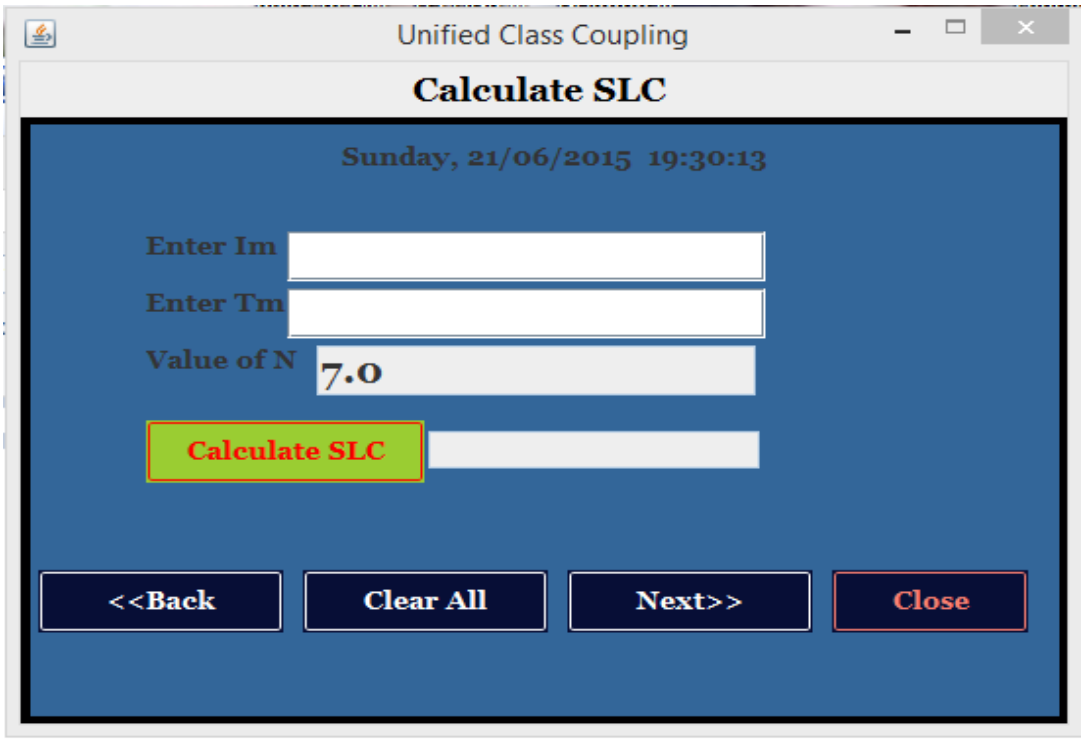



**Figure 5.10: SLC Calculator Interface**

### 5.5.5 UCC Calculator

The interface below display the overall unified class coupling value together with their corresponding coupling classification for the software system under study. From this interface it is easier to tell whether the software system under study is highly coupled or lowly coupled.The view entry report helps to print the overall system report containing the values entered at each and every stage of the assessment of the software system.

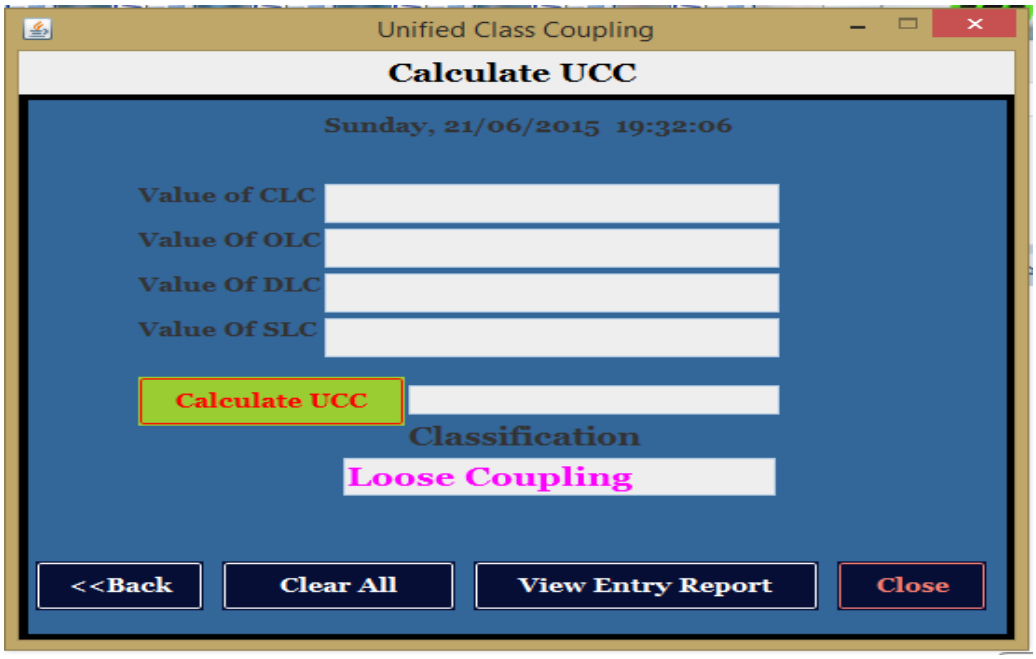



**Figure 5.11: UCC Calculator and Classifier Interface**

# CHAPTER SIX

# 6.0 UNIFIED CLASS COUPLING MODEL VALIDATION

## 6.1 Introduction

This section validates and applies the model to test its application in software development. In this stage 5 applications were selected for study. The applications were from Github.com. Github.com was selected since it has open source software systems. This provided the capability to select software which are freely available and have been studied and used in research. The softwares are also being used by state agencies and therefore they were of good quality.To perform validation three models were selected for use; Li,Leung and Zhang, Hitz and Montazeri, Daisy and Kemerer model. The three models were selected since they had major mechanism of method referencing, method invocation, class aggregation, class inheritance, class parameter return type. The selection therefore provided equal footing and assessment criteria. The goal of validation was to test the relevance and applicability of the research output in assessment of coupling in software system.

## 6.2 Validation Methodology Applied

### 6.2.1 Software Selection Criteria

To qualify for selection for assessment in validation process, the software for study should have had a minimum of 5 classes, objects ,had several interfaces and methods calling each other and making refrences. The software should have the capability of integration with other systems and developed using an object oriented programing language. Similarly the softwares must have been in use for a period of 5 years or more.A software with 5 classess and more would ensure that we have classes to count and assess and possibility of coupling existing within the classes.A software in use for a period of 5 years would ensure that we have industry based software already used and

tested.The software would have undergone through integration testing,corrective maintenance and the software selected for the study must have their source code program freely available for conducting of the experiment.With this cretaria we manage to select 5 software systems to experinmental study in this research.

The selection criteria is based on Bieman and Kang(1999) selection criteria for assessment and testing of the model.According to Bieman and Kang the software used for assessment of the proposed were selected as follows (Bieman & Kang ,1999).

1. A software for assessment should have atleast five classes modeled in it.
2. The software should have been in use for sometime atleast five years of operations
3. A software should have a number of objects,interfaces,methods and interface calling each other.

With regard to open source softwares we anchored the selection on a research conducted by Poshyvanyk *et al* (2009).According to the researchers ,during software change, programmers need to modify the source code of existing software systems. The first step during software change is to identify a part of the source code that needs to be changed. Once the starting point of the change is identified, developers need to identify the other components that need to be changed. Poshyvanyk *et al* (2009) proposed an interactive process in which the programmer or assessor, guided by dependencies among program components (i.e., classes, functions), inspects components one-by-one and identifies the ones that are going to change. This process involves both searching and browsing activities. In order to compute the conceptual coupling of classes, the source code of the software system is converted into a text corpus, where each document contains elements of the implementations of a method (Poshyvanyk *et al*, 2009). Comments and identifiers are extracted from the source code as well as structural information. For this to occur we need open source programs whose codes are freely available to the assessors and programmers (Poshyvanyk *et al*, 2009).

### 6.2.2 Baseline Criteria and Classification

To conduct the validation we used classification and coupling measurement as proposed by Bieman and Kang (1999) as the baseline. This model was selected since it provided capability of variable consideration,methods,attributes and inheritance existing within a class**.** They defined class coupling as an attribute of a class that refers to the connectivity among components of the class (Bieman & Kang 1999). The components include the instance variables, methods defined in a class and methods that are inherited (Bieman & Kang, 1999).

According to Bieman and Kang (1999), a method is said to be connected to an instance variable if the method accesses that instance variable. However, the manner in which an instance variable may be used is different. According to Biemen and Kang (1999) we let ndc (c) be the number of directly connected methods in a class c, nic (c) is the number of indirectly connected methods in a class, and np (c) is the maximum possible number of connections in a class. Then, tight class coupling (tcc) is defined as (Bieman & Kang, 1999).We selected lcc(c) for the study so that we classify all the softwares as either loosely coupled or tightly coupled. From a research conducted by Bieman and Kang loose class coupling (lcc) as defined in Equation 2.2.

These Baseline criteria provided us with a reference to tight class coupling and loose class coupling.From their research they stated that LCC exist for values less than 0.5. According to our research we created highly coupled classes and lowly coupled classes. The LCC as used by Bieman and Kang would be in correlation to High Coupling and LCC would be in correspondence to Low Coupling.

### 6.2.3 Software Applications for Assessment

1. FrontlineSMS is desktop/cloud based software created to lower barriers to positive social change using mobile technology. Its second version is built as a Grails app, which can either run stand-alone as a desktop app, or be bundled as the 'frontlinesms-core-multitenant' plugin that forms the core of Frontline Cloud.

2. Open MCT Desktop: The desktop client is no longer under active development, as our development efforts are now focused on Open MCT for the web and mobile devices.The MCT project was developed at the NASA Ames Research Center for use in spaceflight mission operations, but is equally applicable to any other data monitoring and control application

3. Ushahidi v2: A platform that allows information collection, visualization and interactive mapping, allowing anyone to submit information through text messaging using a mobile phone, email or web form. A Java wrapper for the Ushahidi API also exists. It's simply a Java library that wraps the raw HTTP calls to the Ushahidi APIs. The goal of this SDK is to ease Ushahidi API integration into your applications and to support all the Ushahidi public and admin APIs that are there.

4. NysenateOpen Legislation: It is a web service that delivers legislative information from the New York State Senate and Assembly to the public in near-real time. It is used internally to serve up legislative data for nysenate.gov and the Bluebird-CRM.

5. OpenDuka: It is a project designed by the Open Institute that will provide a freely accessible database of information on Kenyan entities. This information will provide citizens, journalists, and civic activists with a practical and easy-to-use tool to understand the ownership structure of the world they live in.

### 6.2.4 Classification Criteria

The five software studied in this research were classified as either tightly coupled class or loosely coupled class. The classification criterion is based on the number of indirectly connected methods and evaluation as indicated by Bieman and Kang research proposal. The Bieman and Kang equation is used to classify the software as either tightly or loosely coupled.The selected softwares codes were examined and the number of directly connected methods counted and recorded as shown in the table (ndc(c)),the number of indirectly connected methods counted and recorded(nic(c)) and the maximum possible number of connections counted and recorded(npc(c)).The Bieman and Kang Equation 2.2 that states

**lcc(c) = (ndc(c) + nic(c)) / np(c) was applied to the values**.

According to Bieman and Kang loose coupling exist for values betwee 0.0 to 0.499 while tight coupling exist for values between 0.5 to 1 (Bieman and Kang, 1999).The classification evaluation arrived at is as follows

**Table 6.1: Classification Criteria**

| **Software** | **ndc(c)** | **nic(c)** | **npc(c)** | **LCC Value** | **CLASSIFICATION** |
|---|---|---|---|---|---|
| Frontline SMS | 23 | 25 | 132 | 0.364 | Loose coupling |
| OpenMCT Desktop | 18 | 7 | 33 | 0.758 | Tight coupling |
| Open Duka | 13 | 18 | 95 | 0.326 | Loose Coupling |
| Ushahidi $V_2$ | 7 | 8 | 35 | 0.429 | Loose Coupling |
| Nye Senate | 75 | 18 | 154 | 0.604 | Tight Coupling |

### 6.4 Evaluation Process

At this stage we evaluated the 5 software systems by collecting and recording the various values and subjecting them to the Ucc tool created in this research. The main concept of subjecting them to the tool was to get values and classify them as either highly coupled or loosely coupled. The values were arrived on the basis of the proposed model.

### 6.4.1 Class Level Coupling Evaluation

Picking the values of the classes as stated above and applying the formulas, the following table was generated based on the values collected and calculated

**Table 6.2: Class Level Coupling Evaluation**

| Software | n | C | $M_i$ | $M_h$ | $M_a$ | $M_p$ | n*c | CLC |
|---|---|---|---|---|---|---|---|---|
| Frontline | 12 | 8 | 18 | 7 | 18 | 19 | 96 | 0.729 |
| OpenMCT | 9 | 7 | 7 | 18 | 17 | 18 | 63 | 0.907 |
| Open Duka | 8 | 19 | 12 | 18 | 8 | 8 | 152 | 0.979 |
| Ushahidi | 12 | 8 | 16 | 9 | 10 | 11 | 96 | 0.479 |
| NYESenate | 10 | 8 | 18 | 16 | 110 | 13 | 80 | 0.613 |

### 6.4.2 Object Level Coupling Evaluation

Picking the number of objects and applying the object level coupling algorithm and formula, the table below was generated

**Table 6.3: Object Level Coupling Evaluation**

| Software | Message sent | TotaslMessag | OLC |
|---|---|---|---|
| Frontline | 12 | 25 | 0.48 |
| OpenMCT | 9 | 19 | 0.474 |
| Open Duka | 8 | 23 | 0.347 |
| Ushahidi | 12 | 34 | 0.5 |
| NYESenate | 10 | 24 | 0.417 |

### 6.4.3 Dynamic Level Coupling Evaluation

Based on dynamic assessment algorithm discussed and picking the values from the software systems, the table shown below was generated

**Table 6.4: Dynamic Level Coupling Evaluation**

| Software | c | Total Messages | Message sent | Dca | Cs | DKC | SSR | Pac | DLC |
|---|---|---|---|---|---|---|---|---|---|
| Frontline | 8 | 62 | 17 | 0.274 | 7 | 0.875 | 5 | 0.625 | 0.222 |
| OpenMCT | 7 | 60 | 23 | 0.3833 | 6 | 0.857 | 3 | 0.428 | 0.2409 |
| Open Duka | 19 | 46 | 17 | 0.266 | 11 | 0.579 | 8 | 0.421 | 0.0663 |
| Ushahidi | 8 | 46 | 18 | 0.391 | 6 | 0.75 | 7 | 0.865 | 0.251 |
| NYESenate | 8 | 56 | 23 | 0.411 | 5 | 0.625 | 4 | 0.5 | 0.192 |

### 6.4.4 Static Level Coupling Evaluation

In this stage the static algorithm, equation and formula was applied to various data sets in order to assess static level coupling in software systems under study. The table below was generated in the research

**Table 6.5: Static Level Coupling Evaluation**

| Software | Tm | Im | SLC |
|---|---|---|---|
| Frontline | 8 | 5 | 0.625 |
| OpenMCT | 7 | 6 | 0.858 |
| Open Duka | 9 | 5 | 0.5555 |
| Ushahidi | 11 | 9 | 0.818 |
| NYESenate | 6 | 3 | 0.5 |

As shown in table 6.6 we applied the unifed coupling model as stated in Equation 4.3 .We used the individual values got in table 6.2,6.3,6.4 and 6.5 and the constant coefficient $e^{1.090}$ to evelaute the values for Ucc for the software being studied at the validation level of this research.the table shows the classifications as whether high or low based on the values got.

**Table 6.6: Unified Class Coupling Evaluation**

| Software | Ucc | Classification |
|---|---|---|
| Frontline | 0.0624 | High |
| OpenMCT | 0.0919 | High |
| Open Duka | 0.0102 | Low |
| Ushahidi | 0.0711 | High |
| NYESenate | 0.0349 | Low |

## 6.5 Unified Class coupling validation , Observation and Comparison

At this stage we engage in assessment and comparison of pour model against exiting models. To achieve validation we assessed coupling levels of the selected soft waresusing the baseline model,unfied class coupling model and other three models selected that li and henry,Chindaber and Kemerer and Hitz and Montazerimodels proposed in previous research. The values generated are as shown in the table 6.7 below.

**Table 6.7: Model Comparison**

| Software | Bieman and Kang(1999) (Baseline) | | Unified Class Coupling | | Li,leung& Zhang (2013) | | Darcy and Kemere (2005) | | Hitz and Montazeri (2011) | |
|---|---|---|---|---|---|---|---|---|---|---|
| | **Value** | **Level** | **Value** | **Level** | **Value** | **Level** | **Value** | **Level** | **Value** | **Level** |
| FrontlineSMS | 0.364 | Loose | 0.062 | High | 0 | Nil | 6 | High | 3 | Low |
| OpenMCT | 0.758 | Tight | 0.091 | High | 4 | Low | 3 | Low | 3 | High |
| Open Duka | 0.326 | Loose | 0.010 | Low | 7 | High | 4 | High | 4 | High |
| Ushahidi | 0.429 | Loose | 0.071 | High | 0 | Nil | 5 | Low | 3 | Low |
| NYESenate | 0.604 | Tight | 0.0349 | Low | 6 | High | 6 | High | 4 | Low |

From table 6.7 it can be observed that FrotnlineSMS and generated Nil coupling for Li and Henry Model, high for Darcy and Kemerer and low for Hitz and Montazeri. OpenMCT generated low for Li and Henry and Darcy and Kemerer while a high was generated for Hitz and Montazeri. Open Duka generated High for all the three models used in this research. NyeSenate generated high for Li and Henry and Darcy and Kemerer and low for Hitz and Montazeri.

From Table 6.7 Unified class coupling generated a high for Frontline SMS and Ushahidi generated a Nil for Li,Leung and Zhang and low for Hitz and Montazeri model. In the research conducted by Li,Leung and Zhang classes with high data locality are more self sufficient than those with low data locality. This attribute influences external attributes like the class's reuse potential or its testability. Li, Leung and Zhang considers number of disjoint sets of local methods; no two sets intersect; any to methods in the same set share at least one local instance variable; ranging from 0 to N; where N is a positive integer. The chain represents the minimum coupled graph with LCOM=1, while maximum coupling occurs in the complete graph. The obvious generalization of

connectivity leads to the graph theoretic notion of “connectivity of degree k” (k edges must be removed to disconnect the graph) which unfortunately is not easy to determine for higher values of k. The unified coupling model proposed in this research considers both jointed and disjointed methods.The unified coupling model takes into consideration all interacting modules in evaluation of coupling which Li,Leung and Zhang does not consider. Unified coupling model proposed in this research provides a formal improvement of the definition of LCOM, we should get rid of a more semantic flaws in the definition of LCOM: Firstly, the not uncommon design principle to restrict accesses to instance variables to special purpose read/write methods introduces an anomaly of this measure: An otherwise coupled class would yield very high LCOM-values, as all of the “real” methods would yield isolated nodes in the graph, as they do not directly share any instance variable anymore. Secondly, there are many practical cases in which methods exist which do not at all access instance variables (neither directly nor via mere access methods) but are coded entirely in terms of other (more basic) methods of their class. Hitz and Montazeri model involves two approaches to coupling measurement.

1. Coupling between Object (CBO): Is a count of the number of classes to which a class is coupled. It counts each usage as a separate occurrence of coupling. This includes coupling via inheritance.

2. Number of Associations (NAS): is defined as the number of associations of each class. Counted from design documents. Counts repeated invocations as a single occurrence of coupling.If associations are considered stable, no changes are likely to occur and thus Class Coupling will not incur any dependent changes. Stable classes are normally imbedded in the environment of a programming language (“foundation classes”), but they could also be part of an established class library for a certain application domain. Thus, changes in one class require changes in all classes coupled to it. This leads to class level coupling which results from state dependencies between classes during the development life-cycle. This are the reasons for low coupling value.Hitz and Montazeri propose counting of the various instances of interaction but

don’t consider contribution of various objects, methods and associations that exist between classes.In view of this Unified coupling model proposes a measure that take care of interacting modules,methods,

Darcy and Kemerer first define a measure CBO for a class as, a count of the number of non-inheritance related couples with other classes. If the methods of one class use the methods or attributes of the other that implies that the objects of both of the classes are coupled with each other. To improve the modularity of a software the inter coupling between different classes should be kept to a minimum. Beside reusability a high coupling also has a second weakness, a class that is coupled to other classes is susceptible to changes in those classes and as a result it becomes more difficult to maintain and becomes more error-prone. Additionally it is also harder to test a heavily coupled class in isolation. The class becomes so ambiguous that it is quite difficult to understand it. Therefore the number of dependencies should be kept at a minimum. They further refined this definition by saying that CBO for a class is a count of the number of other classes to which it is coupled. Response for class (RFC) The response set (RS) of a class is a set of methods that can potentially be executed in response to a message received by an object of that class. RFC is used to measure the number of different methods that can be executed when an object of that class receives a message (when a method is invoked for that object).

Unified class coupling addresses by providing some other models of assessing the overall coupling in software systems. The Unified class coupling considers aspects of dynamic key coupling, dynamic afferent coupling and percentage active classes that assess dynamic behavior of various software systems.

### 6.6 Validation Discussion

Darcy and Kemerer ( 2005) model measurement describe coupling as any evidence of a method of one object using methods or instance variables of another object. As a

model for coupling, they define CBO (Coupling Between Objects) as proportional to the number of non-inheritance related couples with other classes. Darcy and Kemerer therefore fail to recognize existence of inheritance relationship between objects and classes. In creation and development of large and robust systems, inheritance of objects and characteristics is likely to occur. With recognition of this Darcy and Kemerer model would not only rely on non-inheritance behavior but also inheritance behavior which would also affect static and dynamic behavior characteristics of object oriented software systems. In counting of the number of non-inheritance relationship, this model automatically concludes high coupling in some instances and low coupling in other instnaces as shown in table 6.7. This approach weakness is addressed by the proposed unified class coupling model. The model should have considered methods and attribute interaction between the various objects which affect both static and dynamic behaviors in software systems.

Moreover, it is important to note that Darcy and Kemere rmodel define coupling as an attribute of pairs of objects, but as a model, it is aggregated to the total number of couples that one class has with other classes, thus implicitly assuming that all basic couples are of equal strength. This affects the counting which would directly provide for high coupling in software systems providing high figures as shown in the table 6.7 .This does certainly not hold, e.g., when message passing is inter mingled with direct access to foreign instance variables. This has generally been identified as the worst type of coupling, also in traditional design. But even if only message passing is considered, several authors have distinguished various forms of coupling with different strengths (Darcy & Kemerer, 2005 ). For example, sending messages to objects is considered superior to sending messages to one of an object's components, even if the selection of such a component is accomplished by means of an access message.

The unified model provides a wider alternative of assessing individual objects weight to contribution to coupling, the contribution of all object created within the class and messages being passed between the objects.

As mentioned in the algorithm we should note the fact that in assessing static coupling we considered a set of interacting modules from a set of modules created in the software system. In the unified model we evaluate dynamic coupling by considering Percentage Active Classes ($P_{AC}$) which is defined as the ratio of number of classes sending or receiving at least one method calls from/to another class at runtime to the total number of classes. In this calculation which doesn't recognize the existence but assess the active levels of classes and methods calls which affect coupling in software system. This is the reason why we would end up with highly coupled and lowly coupled software system during validation.

It is not clear whether messages sent to a part of self (i.e., an instance variable of class type) contribute to CBO or not. Strictly adhering to messages to self (or parts thereof) do not alter a foreign object's history and therefore do not constitute coupling. However, it does lead to a certain amount of class level coupling, which cannot be attributed to coupling as defined by (Eder *et al*, 2011).

Li, Leung and Zhang refer to this type of coupling as data abstraction coupling (DAC), measured by the number of instance variables having an abstract data type (Eder *et al*, 2011).By explicitly neglecting inheritance related connections, Darcy and Kemerer exclude from their measure contributions attributed to immediate access to instance variables inherited from super classes, a kind of coupling considered to be among the worst types of coupling (Eder *et al*, 2011).

Darcy and Kemerer also define RFC ( Response For a Class) as the union of the protocol a class offers to its clients and the protocols it requests from other classes. Measuring the total communication potential, this measure is obviously related to coupling and is not independent of CBO (Eder *et al*, 2011).

Li,Leung and Zhang proposes data abstraction coupling (DAC), measured by the number of instance variables having an abstract data type (Eder *et al*, 2011).According

to Li, Leung and Zhang coupling only exist if there is data abstraction between various objects and methods in a software systems.

Li, Leung and Zhang improved the original version of LCOM given in as follows: "LCOM = number of disjoint sets of local methods; no two sets intersect; any to methods in the same set share at least one local instance variable; ranging from 0 to N; where N is a positive integer." The chain represents the minimum coupled graph with LCOM=1, while maximum coupling occurs in the complete graph. The obvious generalization of connectivity leads to the graph theoretic notion of "connectivity of degree k" ( k edges must be removed to disconnect the graph) which is unfortunately not easy to determine for higher values of k. (Eder *et al*, 2011). Using LCOM approach we couldn't establish set of disjoint between the various methods which proved the notion that different objects interact at least for the software systems validated. Li and Henry model would prove existence of coupling between objects if they considered complete sets of interacting modules and objects either through abstraction or interface relationship.

In the software systems assessed in the validation process when assessing unified coupling model we considered methods, objects, classes and attributes and behaviors of the objects as shown in tables 6.2. ,6.3,6.4, 6.5 and 6.7.In calculating coupling as shown in the modeled unified class coupling model, we evaluated the class interaction, objects, methods, interacting modules, dynamic behaviors which eventually contributes to overall coupling in software system. As shown in the table 6.7 Li, Leung and Zhang indicated lack of coupling in software systems with NIL value which shouldn't be the case. If Li, Leung and Zhang provided an option of assessing of coupling by considering both jointed and disjointed characteristics the results would produce high coupling in some software systems under study and low coupling in the other software systems.

Hitz and Montazeri considers coupling between object (CBO) which is a count of the number of classes to which a class is coupled. It counts each usage as a separate

occurrence of coupling. This includes coupling via inheritance. Number of Associations (NAS): is defined as the number of associations of each class .From Hitz and Montazeri if associations are considered stable, no changes are likely to occur and thus class coupling will not incur any dependent changes. Stable classes are normally imbedded in the environment of a programming language ("foundation classes"), but they could also be part of an established class library for a certain application domain.

Thus, changes in one class require changes in all classes coupled to it. This leads to class level coupling which results from state dependencies between classes during the development life-cycle.

As provided in the unified class coupling, a class that is composed of related methods will result in coupling as its methods will use the same set of attributes in their implementation. With this in mind, a class that has single public method will be considered as highly coupled, because all of its attributes will be used by the one public method. In unified coupling we note that coupling exist between classes if one class inherits another class or one class implements an interface, one class has got a method that is invoking method of another class, one class has a method referencing an attribute of another class, one class has got a method having parameter of the type of another class, one class has a method containing a local variable of the type of another class, one class has a method invoking a method having a parameter of the type of another class, methods working on a particular member attribute of class may form group of related functionality, methods using a particulars external resource such as file, database table and memory may form a group of related functionality and methods using same private, protected and public method may also form group of related functionality. This therefore means that unified class coupling model provides a wider means of assessing coupling by considering methods, attributes, functionality and interfaces in software systems.

By putting this into perspective it means that different classes and will have different level coupling. Some will be low while others will be high as shown in the table 6.7.

A further evaluation is provided when assessing dynamic coupling level in software systems i.e Dynamic Key Class ( $D_{KC}$) which defines the ratio of sum of calls sent out from the class and calls received by the class at runtime turned on the total number of static calls sent and received by all the classes. This notion considers the fact that during execution some classes will be static will other classes will be dynamic however; whether static or dynamic some classes will send calls while others will receive calls representing some association set theory between the various classes.
Hitz and Montazeri capability to consider the stated variation would result in a possibility of low coupling value as show in the table 6.6 application during validation process.

## 6.7 Comparison of Unified Model and Previous Research Model

Bieman and Kang in their research proposed two class coupling measures to evaluate the relationship between class coupling and private reuse in the system (Bieman & Kang, 1999). Bieman and Kang defined class coupling to be an attribute of a class that refers to the connectivity among components of the class. The components include the instance variables and the methods defined in a class plus those that are inherited (Bieman & Kang, 1999). The proposed advanced unified model advanced this notion by including pointers, methods and references in evaluation coupling between the various classes and provided the ability to assess the effects of dynamism, objects and static behaviors in assessment of coupling in object oriented systems. This approach of assessment looked in the concept that classes and object may inherit characteristics and therefore provide the ability of importation and exportation of the various components in the software system

In a research conducted by Hitz and Montazeri they apply measurement theory to evaluate and validate any given model. They mentioned that the author must identify the attribute to be measured before any measurement activity (Hitz & Montazeri, 2006). Then a "sufficient" empirical relation system must be established. Having established an empirical relation system, a model m should then map the empirical relation system into an appropriate formal ( or numerical) relation system, preserving the semantics of the empirical relation(s) observed. This model lacked the basic principle of what constitutes the core concepts of object oriented features .The advanced unified model addresses this weakness by providing a concept of inheritance that may affect inter relationship between various objects in OOP system. The advanced unified model also provides a relationship formula that assesses overall class contribution to coupling in software systems, weights determination model and clear description of the various dynamic behavior. The unified formula also recognizes the fact that objects and classes may be exported and imported at different phases of the programming work and therefore recognizes the evaluation of the weights of each of them.

In a research conducted by Briand *et al* (2004) they defined four coupling properties to characterize coupling in a reasonable intuitive and rigorous manner. The four properties are: nonnegative and normalization, null value and maximum value, monotonicity, and merging of unconnected classes (Briand *et al*, 2004). However, these properties are not sufficient to tell that a measure that fulfils them all will be useful; it is likely that a measure that does not fulfill them all is ill-defined. This approach towards measurement ignores the key concept of OOP and coupling in software systems. This weakness is addressed by the proposed unified class model by providing means to assess inheritance in classes and objects, means to assess object exportation and importation in classes and mean to assess their overall net effect in overall coupling of software system. The proposed unified model also checks on overall relationship of object level coupling and static level coupling in software systems. From the unified coupling measurement the indirect coupling and direct coupling is taken care of by determining the number of

pointers and methods created in both classes and objects during program modification and development. This model therefore tries to improve Briand et al approach towards measurement of coupling in software systems.

In a research done ,Chae *et al* (2004) considers two characteristics of classes, which could lead to different coupling values from the general intuition of coupling (Chae *et al*, 2004). The two characteristics are: the patterns of the interactions among the class members (e.g. counting the number of instance variables used by a method) and special methods. The nature of these methods should be reflected to properly measure the cohesiveness (Chae *et al*, 2004).

In order to describe these characteristics in class c, a reference graph $g_r(c)$ is drawn to represent the interaction among the members of class c, and is defined to be an undirected graph $g_r$=(n,a) with (Chae *et al* ,2004):

$$N = V(C) \cup M(C)$$
$$A = \{(m,v) \mid v \in R^*(m) \text{ where } m \in M(C) \text{ and } v \in V(C)\}$$

Where v(c) is a set of instance variables in the class c, m(c) is a set of methods in the class c, and $r^*(m)$ is a set of instance variables directly/indirectly references by m.

Chae *et al* (2004) define glue methods of graph $g_r$, $m_g(g_r)$, as the minimum set of methods without which its reference graph $g_r$ can be divided into disjoint sub-reference graphs. then, they introduce the notion of connectivity factor as: the connectivity factor of a reference graph $g_r$, $f_c(g_r)$, represents the degree of the connectivity among the members, and is defined to be the ratio of the number of the glue methods to the number of normal methods (Chae *et al* , 2004).

The proposed unified model advances this view by introducing the concepts objects coupling, dynamic behavior coupling and their determination factors. The unified model

assesses the effects of class methods on objects and dynamic behavior which in turn would affect coupling and the overall design and maintenance of the software system in future.

In OOP systems allows the implementation of a given operation to be dependent on the object that contains the operation and therefore limits the object ability to acquire features of other objects in OOP programming.

One major principle in software engineering is the design principle “put together what belongs together” (Simon & Parmee, 2008). Three major criteria (point of views) that might lead this grouping are functionality, data orientation, and time orientation. The grouping process, which is called the extraction for reverse engineering, should be tool supported. One instrument for their implementation is distance measure (Simon & Parmee, 2008) .

The work of groupings is strongly connected with the theory of similarity/dissimilarity. Knowing that two given things are similar is not enough, because there are degrees of similarity/dissimilarity in advance. Simon et all proposes that the degree of similarity in coupling measurement can be defined quantitatively if all properties of the object are assigned the same weight ( uniform) (Simon & Parmee, 2008)

Distance based model therefore assigns equal weights to different properties of the object and doesn’t consider class contribution and inheritance characteristics in software engineering. This weakness is addressed by advanced unified coupling since it assesses contribution of classes and their dynamic behaviors in software systems. The unified model assigns different weights to each of the components of assessments and therefore recognizes the unique contribution of each component in the software system.The unified model also considers ignores the concept of similarity since its major core

component is that each object will behave uniquely and contribute uniquely to each of the coupling characteristics in software systems (Simon & Parmee, 2008).

First approach towards measuring object coupling was by proposed by Zhao *et al*,(2014 ). Their approach is based on a dependency model for object-oriented software that consists of a group of dependency graphs. They defined coupling as the degree of relatedness between attributes and modules (method/advice) (Zhao *et al*, 2014). They present, in fact, two ways for measuring object coupling based on inter-attributes (a), inter-modules (m) and module-attribute (ma) dependencies. The first way suggests that each measurement (a, m, ma) works as a field. Therefore, object coupling for a given object A is defined as a 3-tuples Γ(A) = (a, m, ma). They also suggested another way for coupling measurement where each facet could be integrated as a whole with parameters. Object coupling is then expressed as (Zhao *et al*,2014):

$$\Gamma(A) = \begin{cases} 0, & n = 0 \\ \_*\_m, & k = 0 \text{ and } n \neq 0 \\ \_1*\_a + \_2*\_m + \_3*\_ma, & others \end{cases}$$

The Zhao *et al*,(2014) model mainly deals with relatedness of modules and attributes and ignores the net effect of the various classes and objects that are created in object oriented software system. The unified model addresses this limitation by providing means to assess the overall contribution of modules and attributes in classes and objects first before combining their overall contribution in coupling in software systems. This concept is key since the unified model provides the ability to assess the interrelation of all components and how they affect coupling behavior in object oriented software systems.

In a research cnonducted by Sant' Anna et al , the authors proposed a new metric LCOO (Lack of Coupling in Operations) measures the amount of method/advice pairs

that do not access to the same instance variables (Sant'Anna *et al*, 2007). It is an extension of the well-known LCOM (Lack of Coupling in Methods) metric developed by Darcy and Kemerer . This model measures the lack of coupling of a component (class and aspect).

A high LCOO value, according to Sant' Anna et al, indicates disparateness in the functionality provided by the aspect. The definition of LCOO metric is almost operational (Sant'Anna *et al*, 2007). It is not stated whether inherited operations and attributes are included or not, and we have to assume that sets {I } only include attributes of j component $C_1$. It suffers from several problems as stated .The advanced unified model reverses the lack of coupling measurement by proposing the existence of coupling in software systems. The class and object measures are essential in assessing coupling in software systems and this are indicated in various components assessed in objects. The lack of coupling in methods an extension of LCOO is not ideal in measurement of coupling in software systems. This is attributed to the fact that coupling will always exist in OOP systems.

### 6.8 Conclusion

As shown in the validation we propose a unified coupling model for coupling measurement that considers the following aspects:-

1. The number of classes created in software systems
2. The static and dynamic behavior of software systems
3. The set of interacting methods, modules, interfaces and attributes in a software system

The above three aspects are represented by class level coupling, object level coupling,dynamic level coupling and static level coupling

# CHAPTER SEVEN

## 7.0 CONCLUSION AND RECOMENDATIONS

### 7.1 Summary

In this research we have managed to conduct research in various coupling measures that exist in software systems. A research was conducted in Chae et al model, Briand et al model, Xhu and Zhao model, Darcy and Kemerer, Eder et al, Simon et al,Genero et al and Saint Anna et al model. Various model strengths and weakness were discussed and research gap noted to guide the research and provides input for knowledge contribution.

From the research it was noted and concluded that high software coupling would affect maintenance and performance speed of the various software systems. It also noted that loosely or lowly coupled software systems would result in low software maintenance effort and so it was desirable for all software applications to be loosely coupled.

A questionnaire was developed to assist in software development, a host of data was collected from the various business application software developed in JKUAT.A total of 15 soft wares were assessed using the questionnaire developed. During data collection, different aspects of the software system under study were assessed and their net effect on overall software systems recorded. The data collected was subject to Poisson model for assessment and generation of a model that forms a key contribution in this research.

Once data was collected and analyzed a unified class coupling assessment model was proposed and a model generated in diagrammatic form showing the expected inputs ,the evaluation engine and expected output .The model was then translated into set of algorithms and a software tool in java that automates the model was developed in this research. The automated tool was a demonstration of the knowledge and model in a user friendly manner. The interfaces and codes are attached in this research.

To assess and validate the model, a total of 5 software were selected for use in this study. The unified class coupling model was assessed against other previously proposed models namely Li,Leung and Zhang,Hitz and Montazeri,Darcy and Kemerer model. From the validation as proposed UCC model indicated different level of coupling Li,Leung and Zhang indicated no coupling in the software systems,Hitz and Montazeri produced high coupling for all scenarios while Ucc produced both high and low coupling in the software system.

### 7.2 Knowledge Contribution

From the research conducted the key knowledge contribution are as follows

1. This thesis developed a model for assessment of coupling in object oriented software systems
2. The thesis presents an algorithm for assessment of the model developed and presented.
3. This thesis presents a model conceptualizing the model presented in this research.
4. A software tool was developed to automate the calculations of coupling measures

### 7.3 Further Work

From the research it was evident that coupling was majorly done on programs developed in OOP and class concentration. However it is important to note that request for further research can be done on coupling to see on how they affect maintenance and performance of software systems.

Similarly further research can be done effect and assessment of coupling in general design of software systems especially the object oriented software applications.

## REFERENCES


Ågerfalk, P. J., Fitzgerald, B., & Slaughter, S. A. (2009). Introduction to the Special Issue-Flexible and Distributed Information Systems Development: State of the Art and Research Challenges. *Information systems research, 20*(3), 317-328.

Agrawal, A., Chandra, S., & Khan, R. A. (2009, March). An efficient measurement of object oriented design vulnerability. In Availability, Reliability and Security, 2009. ARES'09. International Conference on (pp. 618-623). IEEE.March 16-19.

Al Dallal, J. (2007). A design-based cohesion metric for object-oriented classes. *International Journal of Computer Science and Engineering, 1*(3), 195-200.

Arisholm, E., Briand, L. C., & Foyen, A. (2004). Dynamic coupling measurement for object-oriented software. *Software Engineering, IEEE Transactions on, 30*(8), 491-506.

Arisholm, E., Briand, L. C., & Johannessen, E. B. (2010). A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. *Journal of Systems and Software, 83*(1), 2-17.

Arisholm, E., Gallis, H., Dybă, T., & Sjoberg, D. I. (2007). Evaluating pair programming with respect to system complexity and programmer expertise. *Software Engineering, IEEE Transactions on, 33*(2), 65-86.

Arlow, J., & Neustadt, I. (2005). UML 2 and the unified process: practical object-oriented analysis and design. Pearson Education India.

Arora, K., Singhal, A., & Bansal, A. (2012). Correlation between Various Quality Characteristics for Aspect-Oriented Systems. *International Journal of Computer Applications, 48*(9).

Aryani, A., Perin, F., Lungu, M., Mahmood, A. N., & Nierstrasz, O. (2014). Predicting dependences using domain-based coupling. *Journal of Software: Evolution and Process, 26*(1), 50-76.

Babu, S. (2015). Run Time Coupling Measurement in a Distributed Object Oriented Systems. *Middle-East Journal of Scientific Research, 23*(5), 801-807.

Bartsch, M., & Harrison, R. (2008). An exploratory study of the effect of aspect-oriented programming on maintainability. *Software Quality Journal, 16*(1), 23-44

Bavota, G., De Lucia, A., & Oliveto, R. (2011). Identifying extract class refactoring opportunities using structural and semantic cohesion measures. *Journal of Systems and Software, 84*(3), 397-414.

Edar, X., Palmer, L. E., Bolanos, R., Mircean, C., Fasulo, D., & Wittenberg, G. M. (2011).: an efficient error detection and removal algorithm for next generation sequencing data. *Journal of computational biology, 17*(11), 1549-1560.

Bavota, G., Linares-Vasquez, M., Bernal-Cardenas, C. E., Di Penta, M., Oliveto, R., & Poshyvanyk, D. (2015). The impact of api change-and fault-proneness on the user ratings of android apps. *Software Engineering, IEEE Transactions on, 41*(4), 384-407.

Bettini, C., Brdiczka, O., Henricksen, K., Indulska, J., Nicklas, D., Ranganathan, A., & Riboni, D. (2010). A survey of context modelling and reasoning techniques. *Pervasive and Mobile Computing, 6*(2), 161-180.

Bidve, V. S., & Khare, A. (2012). A survey of coupling measurement in object oriented systems. *International Journal of Advances in Engineering & Technology, 2*(1), 43.

Bird, C., Nagappan, N., Devanbu, P., Gall, H., & Murphy, B. (2009). Does distributed development affect software quality?: an empirical case study of Windows Vista. *Communications of the ACM, 52*(8), 85-93.

Breivold, H. P., Crnkovic, I., & Larsson, M. (2012). A systematic review of software architecture evolution research. *Information and Software Technology, 54*(1), 16-40.

Briand, L. C., Wüst, J., Daly, J. W., & Porter, D. V. (2000). Exploring the relationships between design measures and software quality in object-oriented systems. *Journal of systems and software, 51*(3), 245-273.

Burrows, R., Ferrari, F. C., Garcia, A., & Taïani, F. (2010). An empirical evaluation of coupling metrics on aspect-oriented programs. In Proceedings of the 2010 ICSE Workshop on Emerging Trends in Software Metrics (pp. 53-58). ACM.

Cataldo, M., & Nambiar, S. (2012). The impact of geographic distribution and the nature of technical coupling on the quality of global software development projects. *Journal of software: Evolution and Process, 24*(2), 153-168.

Cataldo, M., Mockus, A., Roberts, J. A., & Herbsleb, J. D. (2009). Software dependencies, work dependencies, and their impact on failures. *Software Engineering, IEEE Transactions on, 35*(6), 864-878.

Céret, E., Dupuy-Chessa, S., & Godet-Bar, G. (2010). Using software metrics in the evaluation of a conceptual component model. In Research Challenges in Information Science (RCIS), 2010 Fourth International Conference on (pp. 507-514). IEEE. , May 19-21.

Chae, H. S., Kwon, Y. R., & Bae, D. H. (2004). Improving cohesion metrics for classes by considering dependent instance variables. *IEEE Transactions on Software Engineering, 30*(11), 826.

Conchúir, E. Ó., Ågerfalk, P. J., Olsson, H. H., & Fitzgerald, B. (2009). Global software development: where are the benefits?. *Communications of the ACM, 52*(8), 127-131.

Counsell, S., Hassoun, Y., Loizou, G., & Najjar, R. (2006). Common refactorings, a dependency graph and some code smells: an empirical study of Java OSS. In Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering (pp. 288-296). ACM. , September 21st .

Cox, G. W., Etzkorn, L. H., & Hughes, W. E. (2006). Cohesion metric for object-oriented systems based on semantic closeness from disambiguity. *Applied Artificial Intelligence, 20*(5), 419-436.

Dang, X. H., & Zhang, S. K. (2005). Study on Relationships between Nodes of Modular Organization in Technology Innovation Networks: *Coupling Components and Conceptual Model [J]*. *China Industrial Economy, 12*(6), 85-91.

Darcy, D. P., & Kemerer, C. F. (2005). OO metrics in practice. *Software, IEEE, 22*(6), 17-19.

De Lucia, A., Oliveto, R., & Vorraro, L. (2008). Using structural and semantic metrics to improve class cohesion. In Software Maintenance, 2008. ICSM 2008. IEEE International Conference on (pp. 27-36). IEEE. , September 28 -October 4.

Dennis, A., Wixom, B. H., & Tegarden, D. (2015). Systems analysis and design: An object-oriented approach with UML. N.Y: John Wiley & Sons.

Etzkorn, L. H., Gholston, S. E., Fortune, J. L., Stein, C. E., Utley, D., Farrington, P. A., & Cox, G. W. (2004). A comparison of cohesion metrics for object-oriented systems. *Information and Software Technology, 46*(10), 677-687.

Farias, K., Garcia, A., & Lucena, C. (2012). Evaluating the impact of aspects on inconsistency detection effort: a controlled experiment (pp. 219-234). Berlin Heidelberg: Springer.

Fong Boh, W., Slaughter, S. A., & Espinosa, J. A. (2007). Learning from experience in software development: A multilevel analysis. *Management Science, 53*(8), 1315-1331.

France, R. B., Ghosh, S., Dinh-Trong, T., & Solberg, A. (2006). Model-driven development using UML 2.0: promises and pitfalls. *Computer, 39*(2), 59-66.

Gélinas, J. F., Badri, M., & Badri, L. (2006). A Cohesion Measure for Aspects. *Journal of object technology, 5*(7), 75-95.

Genero, M., Piattini, M., & Calero, C. (2005). A survey of metrics for UML class diagrams. *Journal of object technology, 4*(9), 59-92.

Gethers, M., & Poshyvanyk, D. (2010). Using relational topic models to capture coupling among classes in object-oriented software systems. In Software Maintenance (ICSM), 2010 IEEE International Conference on (pp. 1-10). IEEE. , September 12-18 .

Gethers, M., Aryani, A., & Poshyvanyk, D. (2012). Combining conceptual and domain-based couplings to detect database and code dependencies. In Source Code Analysis and Manipulation (SCAM), 2012 IEEE 12th International Working Conference on (pp. 144-153). IEEE. , September 23-24 .

Gupta, N., Singh, D., & Sharma, A. (2015). Identifying Effective Software Metrics for Categorical Defect Prediction Using Structural Equation Modeling. In Proceedings of the Third International Symposium on Women in Computing and Informatics (pp. 59-65). ACM. , August 10th .

Hammad, M. (2015). Identifying related commits from software repositories. *International Journal of Computer Applications in Technology, 51*(3), 212-218.

Harrison, R., Counsell, S., & Nithi, R. (2000). Experimental assessment of the effect of inheritance on the maintainability of object-oriented systems. *Journal of Systems and Software, 52*(2), 173-179.

Hongyu Pei Breivold, Muhammad AufeefChauhan and Muhammad Ali Babar( 2010). "What does research say about agile and architecture?." Software Engineering Advances (ICSEA), 2010 Fifth International Conference on. IEEE, 2010.IEEE .August 22-27.

Horgan, J. R., & Mathur, A. P. (1990). Weak mutation is probably strong mutation. Purdue University, West Lafayette, Indiana, Technical Report SERC-TR-83-P.

Jungmayr, S. (2002). Testability measurement and software dependencies. *In Proceedings of the 12th International Workshop on Software Measurement ,25*( 9), 179-202.

Kabaili, H., Keller, R. K., Lustman, F., & Saint-Denis, G. (2000, June). Class cohesion revisited: an empirical study on industrial systems. In Workshop on Quantitative Approaches OO Software Engineering.

Kagdi, H., Gethers, M., & Poshyvanyk, D. (2013). Integrating conceptual and logical couplings for change impact analysis in software. *Empirical Software Engineering, 18*(5), 933-969.

Kalogerakis, E., Christodoulakis, S., & Moumoutzis, N. (2006). Coupling ontologies with graphics content for knowledge driven visualization. In Virtual Reality Conference, 2006 (pp. 43-50). IEEE. March 25-29 .

Kanellopoulos, Y., Dimopulos, T., Tjortjis, C., & Makris, C. (2006). Mining source code elements for comprehending object-oriented systems and evaluating their maintainability. *ACM SIGKDD Explorations Newsletter, 8*(1), 33-40.

Kang, B. K., & Bieman, J. M. (1999). A quantitative framework for software restructuring. *Journal of Software Maintenance, 11*(4), 245-284.

Kayarvizhy, N., & Kanmani, S. (2013). Analyzing and Identifying Potential Areas of Improvement in Object Oriented Metrics. *International Review on Computers and Software (IRECOS), 8*(9), 2213-2220

Kazemi, A., Azizkandi, A. N., Rostampour, A., Haghighi, H., Jamshidi, P., & Shams, F. (2011). Measuring the Conceptual Coupling of Services Using Latent Semantic Indexing. In Services Computing (SCC), 2011 IEEE International Conference on (pp. 504-511). IEEE. July 4-9 .

Kelsen, P. (2003). An information-based view of representational coupling in object-oriented systems. In Fundamental Approaches to Software Engineering (. 216-230). Springer Berlin Heidelberg.

Krishnapriya, V., & Ramar, K. (2010, June). Exploring the Difference between Object Oriented Class Inheritance and Interfaces Using Coupling Measures. In Advances in Computer Engineering (ACE), 2010 International Conference on (pp. 207-211). IEEE. June 20-21.

Kruchten, P. (2004). The rational unified process: an introduction. Addison-Wesley Professional,Indiana.

Kumar, A., Kumar, R., & Grover, P. S. (2007). An evaluation of maintainability of aspect-oriented systems: a practical approach. *International Journal of Computer Science and Security, 1*(2), 1-9.

Lanza, M., & Marinescu, R. (2007). Object-oriented metrics in practice: using software metrics to characterize, evaluate, and improve the design of object-oriented systems. Springer Science & Business Media.

Larman, C. (2005). Applying UML and patterns: an introduction to object-oriented analysis and design and iterative development. Pearson Education India.

Li, B., Sun, X., Leung, H., & Zhang, S. (2013). A survey of code-based change impact analysis techniques. *Software Testing, Verification and Reliability*, *23*(8), 613-646.

Li, S., Shang, J., & Slaughter, S. A. (2010). Why do software firms fail? Capabilities, competitive actions, and firm survival in the software industry from 1995 to 2007. *Information Systems Research, 21*(3), 631-654

Liu, Y., Poshyvanyk, D., Ferenc, R., Gyimóthy, T., & Chrisochoides, N. (2009). Modeling class cohesion as mixtures of latent topics. In Software Maintenance, 2009. ICSM 2009. IEEE International Conference on (pp. 233-242). IEEE. September 20-26.

Losavio, F., Matteo, A., & Morantes, P. (2009). UML Extensions for Aspect Oriented Software Development. *Journal of Object Technology*, *8*(5), 85-131.

Lu, H., Zhou, Y., Xu, B., Leung, H., & Chen, L. (2012). The ability of object-oriented metrics to predict change-proneness: a meta-analysis. *Empirical software engineering*, *17*(3), 200-242.

MacCormack, A., Rusnak, J., & Baldwin, C. Y. (2006). Exploring the structure of complex software designs: *An empirical study of open source and proprietary code. Management Science, 52*(7), 1015-1030.

Marcus, A., Poshyvanyk, D., & Ferenc, R. (2008). Using the conceptual cohesion of classes for fault prediction in object-oriented systems. *Software Engineering, IEEE Transactions on, 34*(2), 287-300.

Marcus, A., Poshyvanyk, D., & Ferenc, R. (2008). Using the conceptual cohesion of classes for fault prediction in object-oriented systems. *Software Engineering, IEEE Transactions on, 34*(2), 287-300.

McConahy, A., Eisenbraun, B., Howison, J., Herbsleb, J. D., & Sliz, P. (2012). Techniques for monitoring runtime architectures of socio-technical ecosystems. In Workshop on Data-Intensive Collaboration in Science and Engineering (CSCW 2012)

McQuillan, J. A., & Power, J. F. (2006). On the application of software metrics to UML models. In Models in Software Engineering (pp. 217-226). Springer Berlin Heidelberg.

Mishra, Suchitra Kumari. "Risk Analysis at Design Level using UML Behavioral Diagrams." PhD diss., 2013.

Nguyen, D. C., Perini, A., & Tonella, P. (2007). A goal-oriented software testing methodology. In Agent-Oriented Software Engineering VIII (pp. 58-72). Springer Berlin Heidelberg

Perepletchikov, M., Ryan, C., Frampton, K., & Tari, Z. (2007). Coupling metrics for predicting maintainability in service-oriented designs. In Software Engineering Conference, 2007. ASWEC 2007. 18th Australian (pp. 329-340). IEEE. April 10-13.

Petrenko, M., & Rajlich, V. (2009, May). Variable granularity for improving precision of impact analysis. In Program Comprehension, 2009. ICPC'09. IEEE 17th International Conference on (pp. 10-19). IEEE. May 17-19 .

Petrenko, M., Poshyvanyk, D., Rajlich, V., & Buchta, J. (2007). Teaching software evolution in open source. *Computer,5* (11), 25-31.

Poshyvanyk, D., & Marcus, A. (2006). The conceptual coupling metrics for object-oriented systems. In null (pp. 469-478). IEEE. Sept. 24 - 27, 2006

Poshyvanyk, D., Marcus, A., Ferenc, R., & Gyimóthy, T. (2009). Using information retrieval based coupling measures for impact analysis. *Empirical software engineering, 14*(1), 5-32.

Pressman, R.S(2005)., "Software Engineering, A practitioner's approach",place of publication: Palgrave Macmillan.

Prikladnicki, R., & Audy, J. L. N. (2010). Process models in the practice of distributed software development: A systematic review of the literature. *Information and Software Technology, 52*(8), 779-791.

Qusef, A., Bavota, G., Oliveto, R., De Lucia, A., & Binkley, D. (2011, September). SCOTCH: test-to-code traceability using slicing and conceptual coupling. In Software Maintenance (ICSM), 2011 27th IEEE International Conference on (pp. 63-72). IEEE.September 25-30.

Ramasubbu, N., & Balan, R. K. (2008, May). Towards governance schemes for distributed software development projects. In Proceedings of the 1st international workshop on Software development governance (pp. 11-14). ACM.

Ressia, J., Gîrba, T., Nierstrasz, O., Perin, F., & Renggli, L. (2014). Talents: an environment for dynamically composing units of reuse. Software: *Practice and Experience, 44*(4), 413-432.

Saied, M. A., Abdeen, H., Benomar, O., & Sahraoui, H. (2015). Could we infer unordered API usage patterns only using the library source code?. In Proceedings of the 2015 IEEE 23rd International Conference on Program Comprehension (pp. 71-81). IEEE Press.May 21-28.

Sangwan, R., Bass, M., Mullick, N., Paulish, D. J., & Kazmeier, J. (2006). Global software development handbook. CRC Press,USA.

Sant'Anna, C., Figueiredo, E., Garcia, A., & Lucena, C. (2007, March). On the Modularity Assessment of Software Architectures: Do my architectural concerns count. In Proc. International Workshop on Aspects in Architecture Descriptions (AARCH. 07), AOSD (Vol. 7).

Sant'Anna, C., Garcia, A., Chavez, C., Lucena, C., & Von Staa, A. (2003, October). On the reuse and maintenance of aspect-oriented software: An assessment framework. In Proceedings of Brazilian symposium on software engineering (pp. 19-34).

Shen, H., Zhang, S., & Zhao, J. (2008, June). An empirical study of maintainability in aspect-oriented system evolution using coupling metrics. In Theoretical Aspects of Software Engineering, 2008. TASE'08. 2nd IFIP/IEEE International Symposium on (pp. 233-236). IEEE. June 17-19.

Simons, C. L., & Parmee, I. C. (2008). User-centered, evolutionary search in conceptual software design. In Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on (pp. 869-876). IEEE.June 1-6 .

Sirbi, K., & Kulkarni, P. J. (2010). Metrics for aspect oriented programming-an empirical study. *International Journal of Computer Applications, 5*(12).

Sirbi, K., & Kulkarni, P. J. (2011). On the Benefit of Quantification in AOP Systems-A Quantitative and a Qualitative Assessment. In Advances in Computer Science and Information Technology (pp. 462-471). Springer Berlin Heidelberg.

Tonella, P., & Ceccato, M. (2004, November). Aspect mining through the formal concept analysis of execution traces. In Reverse Engineering, 2004. Proceedings. 11th Working Conference on (pp. 112-121). IEEE.November 8-12.

Towne, W. B., & Herbsleb, J. D. (2012). Design considerations for online deliberation systems. *Journal of Information Technology & Politics, 9*(1), 97-115.

Tsang, S. L., Clarke, S., & Baniassad, E. (2004, May). An evaluation of aspect-oriented programming for java-based real-time systems development. In Object-Oriented Real-Time Distributed Computing, 2004. Proceedings. Seventh IEEE International Symposium on (pp. 291-300). IEEE. May 14th

Újházi, B., Ferenc, R., Poshyvanyk, D., & Gyimóthy, T. (2010). New conceptual coupling and cohesion metrics for object-oriented systems. In Source Code Analysis and Manipulation (SCAM), 2010 10th IEEE Working Conference on (pp. 33-42). IEEE. Septmber 12-13.

Vanderfeesten, I., Cardoso, J., Mendling, J., Reijers, H. A., & Van der Aalst, W. M. P. (2007). Quality metrics for business process models. *BPM and Workflow handbo*ok, *144*, 179-190.

Voas & Zhang, 2009, J., & Zhang, J. (2009). Cloud computing: new wine or just a new bottle?. *IT Professional Magazine, 11*(2), 15.

Wah, K. S. (2000). A theoretical study of fault coupling. *Software testing, verification and reliability, 10*(1), 3-45.

Ward, J., & Daniel, E. (2006). Benefits management: Delivering value from IS & IT investments (p. 418). Chichester: John Wiley & Sons.

Wheeldon, R., & Counsell, S. (2003). Power law distributions in class relationships. In Source Code Analysis and Manipulation, 2003. Proceedings. Third IEEE International Workshop on (pp. 45-54). IEEE. September 26-27 .

Wilkie, F. G., & Kitchenham, B. A. (2000). Coupling measures and change ripples in C++ application software. *Journal of Systems and Software, 52*(2), 157-164

Yadav, A., & Khan, R. A. (2012). Impact of Cohesion on Reliability. *Journal of Information and operations Management, 3*(1).

Yang, Y., Zhou, Y., Lu, H., Chen, L., Chen, Z., Xu, B., ... & Zhang, Z. (2015). Are slice-based cohesion metrics actually useful in effort-aware post-release fault-proneness prediction? An empirical study. *Software Engineering, IEEE Transactions on, 41*(4), 331-357.

Zhao, J. (2004, September). Measuring coupling in aspect-oriented systems. In 10th International Software Metrics Symposium (Metrics 04).

Zhao, Y., Yang, Y., Lu, H., Liu, J., Leung, H., Wu, Y., ... & Xu, B. (2016). Understanding the value of considering client usage context in package cohesion for fault-proneness prediction. Automated Software Engineering, 1-61.

Zhou, Y., Leung, H., & Xu, B. (2009). Examining the potentially confounding effect of class size on the associations between object-oriented metrics and change-proneness. *Software Engineering, IEEE Transactions on*, *35*(5), 607-623.

Zhou, Y., Yang, Y., Xu, B., Leung, H., & Zhou, X. (2014). Source code size estimation approaches for object-oriented systems from UML class diagrams: A comparative study. *Information and Software Technology*, *56*(2), 220-237.

## APENDICES

Data Collection Experinment Sheet

### GENERAL SECTION(Tick Appropriately)

Please Specify the names of software under assessment

……

Please specify the category or business area the software is used in (i.e. Sacco, Banking etc)

……..

1.What best describes your occupation in the software development team

Software Analyst ☐ Software Programmer ☐ Project Manager ☐ Software Tester ☐

Other Specify ☐

2.How many Years have you been in software Development

1-3 ☐ 4-8 ☐ 9 and Above ☐

4. Do you have any formal training in software development or any field related to software development Yes ☐ No ☐ N/A ☐

If Yes Specify Computer Science ☐ IT ☐ Electricals and Electronics ☐ Programming ☐

Are you familiar with any coupling measurement Model Yes ☐ No ☐

The software to be assessed is developed with which programming language……..……..

How many classes are created within the software to be developed……..

**SOFTWARE ASSESSMENT SECTION(Fill in Appropriately)**

1. **Class Level Coupling Assessment Section**

***Instructions*:** In this section the software assessor is required to modify coupling between the various classes in software under assessment and record the accrued effects in the table below

| No Of Exported Classes | No of Imported Classes | No of Attributes Affected | New References Created | New Pointers Created | Aggregation Created | No of Methods Invoked | No Of New Inherited Classes |
|---|---|---|---|---|---|---|---|
| | | | | | | | |

2. **Object Level Coupling Assessment Section**

***Instructions:*** In this section the software assessor I required to modify coupling between the various objects in software under assessment and record the accrued effects in the table below

| **No Of Exported Classes** | **No of Imported Classes** | **No of Attributes Affected** | **New References Created** | **New Pointers Created** | **Aggregation Created** | **No of Methods Invoked** | **No Of New Inherited Classes** |
|---|---|---|---|---|---|---|---|
| | | | | | | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | | | |

### 3. Static Level Coupling Assessment Section

***Instructions***: In this section the software assessor I required to modify coupling between the various static methods in software under assessment and record the accrued effects in the table below

| No Of Exported Classes | No of Imported Classes | No of Attributes Affected | New References Created | New Pointers Created | Aggregation Created | No of Methods Invoked | No Of New Inherited Classes |
|---|---|---|---|---|---|---|---|
| | | | | | | | |

### 4. Dynamic Level Coupling Assessment Section

***Instructions:*** In this section the software assessor I required to modify coupling between the various Dynamic methods in software under assessment and record the accrued effects in the table below

| No Of Exported Classes | No of Imported Classes | No of Attributes Affected | New References Created | New Pointers Created | Aggregation Created | No of Methods Invoked | No Of New Inherited Classes |
|---|---|---|---|---|---|---|---|
| | | | | | | | |

Any observations and comments from the assessment …….