The International Journal Of Engineering And Science (IJES)
|| Volume || 4 || Issue || 10 || Pages || PP -01-10|| 2015 ||
ISSN (e): 2319 – 1813 ISSN (p): 2319 – 1805

Coupling Measures for Object Oriented Software Systems- A State-of-the-Art Review.

Calvins Otieno [1], George Okeyo[2] and Stephen Kimani[3]
[1]*Computing Department, School of Computing and Information Technology, Jomo Kenyatta University Of Agriculture and Technology, Company, Nairobi, Kenya*

----------**ABSTRACT**----------
Coupling describes the interrelationship between the various functionalities in a software system. High coupling is a considered as a characteristic of poorly designed system while Low coupling is considered as a characteristic of a well-designed system. This paper presents a review of existing coupling measures and classifies them on the basis of class consideration, object considerations and static and dynamic behavior consideration.

Keywords: *Coupling, Framework, Measure, Class, Object, Object Oriented.*

Date of Submission: 17 September 2015 Date of Accepted: 03 October 2015

I. Introduction

Coupling measures capture the degree of interaction and relationships among source code elements, such as classes, methods, and attributes in object-oriented software systems. One of the main goals behind object oriented analysis and design is to implement a software system where classes have low coupling among them. These class properties facilitate comprehension activities, testing efforts, reuse, and maintenance tasks [3]. Coupling affects 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 constructers is a desirable characteristic of an object oriented system [3]. Coupling can also be described in terms of 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 [11].

Many frameworks have been proposed to measure the coupling and cohesion to predict the fault-proneness and maintainability of software systems [5]. However, few studies have been ON done using coupling to measure reusability of software components because of their limitations and the difficulties to evaluate [5].
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 [17].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 [11], assessing the fault-proneness of classes [2], fault prediction [2], re-modularization, identifying of software components [17], design patterns [11] , assessing software quality[17], etc. In general, one of the goals of the software designers is to keep the coupling in an OO system as low as possible [12]. 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 [9]. 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 object oriented software attributes. Many frameworks have been proposed in the last several years to measure class coupling in object-oriented systems. Class coupling (more specifically, functional coupling) is defined as the degree of relatedness between members of a class [14]. In object oriented software systems, a class should represent a single logical concept, and not to be a collection of miscellaneous features. Object oriented analysis and design methods promote a modular design by creating high coupled classes [14]. However, improper assignment of responsibilities in the design phase can produce low coupled classes with unrelated members.

The purpose of this paper is to provide state of the art review of various coupling measures, critically evaluating each measure and providing a better way of classifying the measures on the basis of class, objects, static and dynamic behavior. The key contribution of this paper is the classification model of coupling on the basis of classes, objects, static and dynamic review. To contribute the knowledge we adopted a mechanism, consisting of the following.

(i) Methods share data (public attributes),
(ii) Methods references attributes
(iii) Methods invokes other methods
(iv) Methods receive pointer to other methods
(v) Class is a type method parameter or return type
(vi) Class is a type of method local variable
(vii) Class is a type parameter of a method invoked from within another method
(viii) Class is ancestor of another class

The structure adopted for this paper is statement of the methodology adopted for the paper, review of the related work, critique of the various measures in related work, presentation of conclusion of the state of the art.

II. Methodology

To conduct state of the art review, we selected papers for review that were done within the last 5 years for review. The objective of selection of the papers done within the last five years was to ensure that we get the recent state of the art paper that would provide relevant information. The paper was then to be classified as either conference paper or journal papers. For a paper to be considered for review it had to be indexed by Google scholar, SCI, Cite Seer, directory of open access journals.

The related work was then grouped according to the mechanism that they address as stated in the introduction. From the mechanism they are then classified according to class based, object based, static based and dynamic based.

A critique citing weaknesses and strengths of each journal coupling measure is then presented in the paper and a conclusion drawn in the paper on the basis of observations made on the reviews.

III. Coupling Measures

3.1 Introduction

Coupling measurement is a focus of study for many of the software professionals from 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 [3].

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 [5] .

Class coupling (more specifically, functional coupling) is defined as the degree of relatedness between members of a class [14]. In object oriented software systems, a class should represent a single logical concept, and not to be a collection of miscellaneous features. Object oriented analysis and design methods promote a modular design by creating lowly coupled classes [14] .

One of the main goals behind Object Oriented analysis and design is to implement a software system where classes have low coupling among them. These class properties facilitate comprehension activities, testing efforts, reuse, and maintenance tasks [3].

3.2 Package Coupling Measures

For object-oriented systems, most of the coupling framework have been defined up to class level and only a few framework exist for measurement of coupling at the higher levels of abstraction in object-oriented systems[20] . Other work related to packages or other higher abstraction levels has been carried out in Hongyu et al, 2010)[20].Lee et al. proposed information-based coupling(ICP), a coupling measure for a set of classes based on information flow through method invocations within classes. They defined coupling of a set of classes as the sum of the couplings of the classes in the set [20]. Martin proposed various package level coupling measures such as package coupling, He defined package coupling as the number of classes from other packages that depend on the classes within the package and independent package coupling as the number of classes from other packages that the classes within the package depend upon [21]. 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 [21]. Similarly, Tagoug proposed a coupling measure for subjects . 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 framework for object-oriented systems on the basis of difference between cohesion and couplings of elements of a system [21] . But, he 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 [21].

These relationships among elements of different packages present at the same hierarchical level give rise to the inter-package coupling. 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 [13]. 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.

3.3 Class Coupling Measures

Hitz and Montazeri, described Class level coupling (CLC) as coupling resulting from state dependencies between two classes in a system during the development lifecycle [6]. According to Hitz and Montazeri, 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 [6]. 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), sc is the accessed class (server class). The factors determining the strength of CLC between client class c and server class are [6]:

(i) 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.

(ii) 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.

In another research Bieman and Kang proposed two class coupling measures to evaluate the relationship between class coupling and private reuse in the system [1]. Bieman and Kang research focuses on two important criteria of coupling measure;

1 the interaction pattern

2 The special method.

Bieman and Kang defined class coupling as an attribute of a class that refers to the connectivity among components of the class[1]. The components include the instance variables, methods defined in a class and methods that are inherited [1].

According to Bieman and Kang, a method is said to be connected to an instance variable if the method accesses that instance variable [1]. 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 [1].

According to Biemen and Kang 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 and Kang, 2010)

tcc(c) = ndc(c) / np(c) ……*(1)*

and loose class coupling (lcc) is defined as:

lcc(c) = (ndc(c) + nic(c)) / np(c) …..*(2)*

This framework focuses on two important criteria of coupling measure; the interaction pattern and the special method. The strongest part in this framework is to consider the interaction between pattern methods.

In another research Chae et al considers two characteristics of classes, which could lead to different coupling values from the general intuition of coupling [4]. 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[4] .

According to Chae et al , 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 [4]: The relationship for evaluating coupling in the frameworks is given as follows

$N = V(C) \cup M(C)$ ……*(3)*

$A = \{(M,V)!V \in R(M)\}$ …… *(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 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[4].

In their research Briand et al 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 [2] . However, these properties are not sufficient to tell that a measure that fulfills 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 Rc. We say that $R_c$is maximal, if all possible relationships within class c are presented. We say Rc is maximal if R_c is maximal $\forall c \in c$ [2] .

A research conducted by Harrison et al defines coupling between classes (CBC) as a count of the number of classes to which a class is coupled [15]. 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[15] .

A further research done by Harrison et al (considers the Number of Associations (NAS) in a class. In this research three hypotheses related to coupling are investigated by the authors are [15]:

(i) H1: As inter-class coupling increases, the understandability of a class decreases. This hypothesis is rejected by authors.
(ii) H2: As inter-class coupling increases, the total number of errors found in a class increases. This hypothesis is rejected by authors.
(iii) H3: 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

(i) Software Understandability (SU)
(ii) Number of known errors (KE)
(iii) 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 authors [15].

In this research it is noted that source lines of codes framework measures the number of physical lines of active code, that is, no blank or commented lines code[13]. 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 [15] .

In another research as proposed by simon et al , they advances one major principle in software engineering; design principle, "put together what belongs together" [16]. 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 [16].

In the research the work of groupings is strongly connected with the theory of similarity/dissimilarity. Simon et al 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) [16].

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.

3.4 Object Coupling Measures

In a research by Hongyu et al the researchers proposes 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 [21] :

(i) ***Number of children approach:*** -Number of children framework defines number of sub-classes subordinated to a class in the class hierarchy. This framework measures how many sub-classes are going to inherit the methods of the parent class. Number of children framework relates to the notion of

scope of properties. If number of children framework grows it means reuse increases. On the other hand, as number of children framework increases, the amount of testing will also increase because more children in a class indicate more responsibility. So, number of children framework represents the effort required to test the class, reuse and maintain.

(ii) ***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.

(iii) ***Number of Dependencies In: -***The Number of Dependencies In framework is defined as the number of classes that depend on a given class. This framework 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.

(iv) **Number of Dependencies In *Number of Dependencies Out:-***The Number of Dependencies Out framework is defined as the number of classes on which a given class depends. When the framework value is minimum the class can function independently.

(v) **Number of Dependencies Out *Number of Association:* -** The Number of Association per Class framework 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.

(vi) ***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.

In a research Chidamber and Kemerer they transforms the definition by Wand and Weber, Chidamber and Kemerer conclude "any evidence of a method of one object using methods or instance variables of another object constitutes coupling [22].As a framework 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 [22]. Moreover, we may note that coupling is defined as an attribute of pairs of objects, but as a framework, 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.

In a research by Eder et al they identify the three dimensions of coupling are derived [7].

(i) 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 [7].

(ii) 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.

(iii) Inheritance coupling: Two classes' c and c¢ are inheritance coupled, if one class is an ancestor of the other.

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

(i) Static: can only calculate design time properties of an application.

(ii) Dynamic: used to calculate actual runtime properties of an application.

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

(i) Class level coupling (CLC): Only invocations from one method to another are considered.

(ii) 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 framework authors used ROOM design charts, cyclomatic complexity, operation complexity is calculated from ROOM charts [23].

Authors explained export and import object coupling with context, description, formula and impact on design quality attributes [23] .

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 [23] .

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 [23].

3.5 Dynamic and Static Coupling Measures

The framework described by Erik considered following points regarding coupling measurement [24].

(i) Dynamic behavior of software can be precisely inferred from run time information.
(ii) 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 .

(i) Mapping: object or class
(ii) Direction: import or export
(iii) 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 [24]

(i) Scenario: a specific sequence of interactions between the objects.
(ii) 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 [24].

(i) 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 [24] .
(ii) 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 [24].
(iii) Strength of Coupling: The strength of coupling quantifies the amount of association between the roles.

In a research conducted by Briand et al 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 classification of dynamic coupling measures [2].

(i) Entity of measurement: The entity of measurement may be a class or an object.
(ii) Granularity: The granularity can be class level or object level.
(iii) Scope: The objects and classes are to be accounted for measurement.

The authors captured situations for import and export coupling.

(i) Dynamic messages: Total number of distinct messages sent from one object to other objects and vice versa, within the scope considered.
(ii) Distinct method invocations: Number of distinct methods invoked by each method in each object.
(iii) 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 [2] .

(i) The type of connection: What items (attribute, method or class) constitutes coupling .
(ii) The locus of impact: To decide whether to count import or export coupling .
(iii) 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 .
(iv) 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) .
(v) Direct or indirect coupling: To decide whether to count direct or indirect coupling. For example, if a method m1 invokes a method m2, which in turn invokes a method m3, we can say that m1 indirectly invokes m3. Methods m1 and m3 are indirectly connected .
(vi) Inheritance: inheritance-based vs. non-inheritance-based coupling, and how to account for polymorphism, and how to assign attributes and methods to classes .

A varied approach for coupling measurement is proposed by Sant'Anna et al [25] .The authors proposed a new framework Lack of Coupling in Operations(LCOO) which measures the amount of method/advice pairs that do not access to the same instance variables [25]. It is an extension of the well-known LCOM (Lack of Coupling in Methods) framework developed by Chidamber and Kemmerer. This framework 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 framework is almost operational [25]. 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 C1.

Definition of this framework is based on AspectJ-like programming languages whose approach to software development do not strictly follow object oriented software design [25] .

IV. Discussion

Bieman and Kangs research focuses on two important criteria of coupling measure; the interaction pattern and the special method. The strongest part in this framework is to consider the interaction between pattern methods. However, the connectivity factor could generate misleading information leading to poor recognition of the interaction pattern [18]. 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 [18]. 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 framework 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 [18].

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 framework doesn't consider the number of sub components and method invocation which has a direct influence on coupling in classes. The montazeri framework counts edges and doesn't consider the objects that can be created in software system. It's therefore not an appropriate framework 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 fulfills 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. The following are the gaps created by the four proposals approach by Briand et al framework.

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 m1 and m2 then the representation of m1→m2 is equivalent to m2→m1.

The Chae et al framework considers mostly interaction among the various class components and represents this interaction using graphs[4] .However drawing the graph doesn't take into consideration the various instance variables that should be adopted when developing a system .Similarly the graph drawn cannot clearly indicate the difference between imported and exported classes, methods .

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 . 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 .

The framework 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 [19];

a. Component coupling. There are four degrees of component coupling between classes (listed below from strongest to weakest) [17] .
   (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, this is not true since for object oriented systems coupling between components will always exist.
b. Inheritance coupling. There are three degrees of weaknesses of coupling as regards Zhao and Xu inheritance coupling and they are ;
   (i) Inheritance modification: Inheriting class changes at least one inherited method in a manner that violates some predefined "good practice" rules. Eder et al 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. 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 does not strictly follow object oriented software design.

V. Conclusion

In this paper a review has been conducted in the various coupling measures. As part of review we propose classification of coupling measures as class based, object based, dynamic based and static based. A key contribution of the state of the art review is new structure of classification of coupling measures on the basis of class, objects, dynamic and static behavior. A mechanism structure has also been proposed to be used with the list of items to consider proposed. A critique has been presented detailing the weaknesses of each measure and we request for further research to address the various weaknesses as stated in this review paper.

References

[1] Biamen Keller, and Kiang, F., "Coupling as Changeability Indicator in Object-Oriented Systems", Proceedings of the Fifth European Conference on Software Maintenance and Reengineering (CSMR 2010), Estoril Coast (Lisbon), Portugal, 2010.

[2] Briand, L.C., Daly, J., Porter, V., and Wuest, J., "A unified framework for coupling measurement in object-oriented systems", Empirical Software Engineering, 3 (1), 2008.

[3] Chae, H.S. and Bae, D.H., "A coupling measure for object-oriented classes", Software Practice and Experience, No. 30, 2011.

[4] Chae, H.S., Kwon, Y.R., and Bae, D.H., "Improving coupling frameworks for classes by considering dependent instance variables", IEEE Transactions on Software Engineering, vol. 30, no. 11, 2010.

[5] De Lucia, A., Oliveto, R., and Vorraro, L., "Using structural and semantic frameworks to improve class coupling", International Conference on Software Maintenance, 2008.

[6] Denys Hintz and Andrian Mozeri, "The Conceptual Coupling Metrics for Object-Oriented Systems,"ICSM '06 Proceedings of the 22nd IEEE International Conference on Software Maintenance, 2011.

[7] Eder H., Yamasaki, K., Yamada, H., and Noda, M.T., "A proposal of class coupling frameworks using sizes of coupled parts", Knowledge-Based Sof. Engineering, T. Welzer et al. (Eds) IOS Press, 2011

[8] Gelinas J.F, Badri M. , "A Cohesion Measure for Aspects", in Journal of Object Technology, vol. 5, no. 7, pp. 97 – 114, September - October 2011.

[9] Henderson-Sellers, B., "Object-Oriented Frameworks Measures of Complexity", Prentice-Hall, 2011.

[10] Jungmayr, S., "Testability Measurement and Software Dependencies", Proceedings of the 12th International, Workshop on Software Measurement, October 2012

[11] Larman, G., "Applying UML and Design Patterns, An introduction to object-oriented analysis and design and the unified process", Prentice Hall, 2010

[12] Marcus, A., and Poshyvanyk, D., "The conceptual coupling of classes", Proc. 21th IEEE International Conference on Software Maintenance, September 2009

[13] Patidhar et al. International Journal of Advanced Research in Computer Science and Software Engineering 3(3), March - 2013, pp. 517-521 © 2013, Page 518

[14] Pressman, R.S., "Software Engineering, A practitioner's approach", McGraw Hill, 2005

[15] R. Harrison, S. Counsell, R. Nithi "A Cohesion Measure for Aspects", in Journal of Object Technology, vol. 5, no. 7, pp. 97 – 114, September - October 2011.

[16] Simon Allier, St´ephaneVaucher, Bruno Dufour, and HouariSahraoui, 2010 Working Conference on Source Code Analysis and Manipulation,IEEE.

[17] Simon, C., Cox, G., and Etzkorn, L., "Exploring the relationship between coupling and complexity", Journal of Computer Science. 1 (2), 2010.

[18] Voas, J.M., "PIE: A dynamic failure-based technique", IEEE TSE, 18(8), August 2012.

[19] Zhao .J and Xu. B, "Measuring Aspect Cohesion", Proc.International Conference on Fundamental Approaches toSoftware Engineering (FASE'2004), LNCS 2984, pp.54-68 , Springer-Verlag, Barcelona, Spain, March 29-31,2009

[20] Kabaili, H., Keller, R.K., Lustman, F., and Saint-Denis, G., "Class Coupling Revisited: An Empirical Study on Industrial Systems", Workshop on Quantitative Approaches OO Software Engineering, 2010.

[21] Hongyu Pei Breivold, Muhammad Aufeef Chauhan and Muhammad Ali Babar, 2010 Asia Pacific Software Engineering Conference, IEEE.

[22] A Chidamber and P Kemerer., "The conceptual coupling of classes", Proc. 21th IEEE International Conference on Software Maintenance, September 2011.

[23] Sherif M. Yacoub, Hany H. Ammar, and Tom Robinson "Coupling as Changeability Indicator in Object-Oriented Systems", Proceedings of the Fifth European Conference on Software Maintenance and Reengineering, Estoril Coast (Lisbon), Portugal, 2011.

[24] Erik Arisholm. Maintenance Metrics for the Object Oriented Paradigm. In Proc. 1st Int. Software Metrics Symp., Los Alamitos,2011

[25] Cláudio Sant'Anna, Eduardo Figueiredo, Alessandro Garcia, Carlos J. P. Lucena, "On theModularity Assessment of Software Architectures: Do my architectural concerns count?", 6th International Workshop on Aspect-Oriented Software Development(AOSD • 07), Vancouver, British Columbia, March , 2010.

Calvins Otieno is a PhD student in the Department of Computing at the Jomo Kenyatta University of….. He has a B.Sc. in Computer Technology, Masters in Software Engineering. His research interest is in the fields of software engineering and software quality systems. He currently teaches in IT department in the same university.

Dr.George Okeyo Onyango is the current Chairman and HOD Computing at JKUAT. He has a PhD in Computer Science from University of Ulster, MSc Information Systems from University of Nairobi and BSc Mathematics and Computer Science from JKUAT. His research intrest are in the fields of object oriented technologies,semantic web and pervasive computing.

Dr. Stephen Kimani is the current director School of Computing and Information Technology of the Jomo Kenyatta University of Agriculture and Technology in Kenya. He has previously been a researcher at CSIRO (Australia) and Sapienza University of Rome (Italy). He has the following academic qualifications: BSc in Mathematics and Computer Science (JKUAT); MSc in Advanced Computing (University of Bristol, UK); PhD in Computer Engineering (Sapienza University of Rome, Italy).