The International Journal Of Engineering And Science (IJES)
|| Volume || 5 || Issue || 2 || Pages || PP -37- 44 || 2016 ||
ISSN (e): 2319 – 1813 ISSN (p): 2319 – 1805

A Unified Coupling Model for Coupling Measurement in Object Oriented Software Systems.

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 is the extent to which the functions performed by a subsystem are related. Low coupling is a characteristic of a well-designed subcomponent. The problem noted for research in this paper is lack of standardization of measures, ambiguity in definition of measures and poor conceptual links between coupling components. The key objective of this research paper is to propose a model for coupling measurement in object oriented software system. A review of related work is conducted on coupling frameworks. In this research data was collected from various software systems, analyzed using a Poisson distribution model and unified coupling model generated. From the model an algorithm was developed for assessing coupling in software systems. In conclusion coupling model of five components; object level coupling, static level coupling, dynamic and static level coupling is developed an algorithm proposed to support coupling measurement in object oriented software systems. In the unified model developed a coupling value of 0.049 and below indicates low coupling while a coupling value of 0.05 to 0.1 indicates high coupling in the software system.
Keywords: *Coupling, Model, Framework, Object Oriented Measurement, Measure.*

Date of Submission: 29 September 2015 Date of Accepted: 01 March 2016

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.
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 done using coupling to measure reusability of software components.[5].
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 object oriented design 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.
Coupling is a more complex software attribute in object oriented systems but our understanding about coupling measurement factors is poor [8]. There is no standardization for expressing coupling measures; many measures are not operationally defined i.e. there is some ambiguity in their definitions [8]. As a result, it is difficult to understand how different measures relate to one other and what their potential use is. All above aspects ultimately shapes the need of detailed study of coupling measurement in object-oriented systems
Briand et al in their research notes that there is little understanding of the motivation and empirical hypotheses behind many of these new measures [2]. 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 [2].

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 [8].

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

This paper proposes a unified model that helps in coupling measurement. The model developed in this paper provides weights and evaluation function to determine coupling levels in software system. We believe the model would be effective in measurement since it looks into class coupling level, dynamic coupling level, static coupling level and object coupling level. The proposed model has classification criteria that would help reviewers and assessors know whether software is highly coupled or loosely coupled. To enhance the applicability of the model we have developed and proposed an algorithm to outline a

The remainder of this paper is organized as follows review of related work is conducted and critique of the previous model and measures provided, a developed unified model is presented, an evaluation of the model is conducted and results of the evaluation presented, a discussion of the results and model presented and a conclusion of the model and the paper together with request for further research proposed.

II. Related Work.

In a research by Hitz and Montazeri they described Class Level Coupling (CLC) as coupling resulting from state dependencies between two classes in a system during the development lifecycle [6]. According to their research, 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 be 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 a research conducted by Bieman and Kang [1] they 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.

According to Biemen and Kang[1] 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 [1]

tcc(c) = ndc(c) / np(c)

and loose class coupling (lcc) is defined as:

lcc(c) = (ndc(c) + nic(c)) / np(c)

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. However, the connectivity factor could generate misleading information leading to poor recognition of the interaction pattern [18].

In another research Chae et al [4] 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 Chae et al [4], 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

$$N = V(C) \cup M(C)$$
$$A = \{(M,V) ! V \in R(M)\}$$

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 [4] 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.

Briand et al define four coupling properties to characterize coupling in a reasonable intuitive and rigorous manner [2]. 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].

In the very first research conducted by Briand et al in 2005, 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 framework. On the other hand, it is stated implicitly that we have to validate the frameworks that we are going to use for assessing or quantifying the attributes of software [2].Briand et al added some criteria to compare various approaches in order to be in the same perspective [2]. The five criteria of the framework 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 [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)

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, which 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].

Further research on coupling by Zhao and Xu [19] 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 [19];

III. Unified Coupling Model

To generate unified coupling model total of 15 soft wares were studied and analyzed. The data from the soft wares were analyzed in a Poisson model and parametric table below containing beta values generated. The beta values formed the input to the model as shown in table 1 below.. Poisson model was used to analyses the count data produced in the research. The classes for each software under study was modified and the results recorded in a table. The values were then fed into a Poisson mat lab distribution model and the values generated in the table as shown below

Table 1 Parametric 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 1 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 poison model to be used to build the predictive framework in this research. From the table in can be noted that the overall β=1.090, CLC β=0.64, OLC β=-10112, SLC β=1.070, DLC β=0.045, CLC*OLC β=0.156,CLC*SLC β=-0.235

From the table 4 the β can therefore be used to form the poison formula given by

$$Log(Y) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \beta_4 X_4 + \beta_5 X_5 + \beta_6 X_6 \qquad \text{………….(1)}$$

On substituting the values we have

$$Log_e(unified\ coupling) = 1.090 + 0.644\ CLC - 1.112\ OLC + 1.070\ SLC + 0.45\ DLC + 0.156\ CLC * OLC - 0.235\ CLC * SLC \qquad \text{……………(2)}$$

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

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

……………(3)

The equation 3 above represents the proposed model for this research. The model enables us to evaluate the value for unified coupling in software systems. To construct the model we used the parametric estimates value as shown in table 1.

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 betas exponential values were generated as a result figures obtained from the parametric estimates table.

To evaluate the various components we proposed the following approaches to help us determine the static coupling values, dynamic coupling values, object coupling values and class coupling values

3.1 Class Level Coupling

The algorithm to be adopted will have components as follows

- C_i : is an instance of a class
- C_j : is the set of classes invoking the methods or inheriting methods of other classes
- M_i: the number of methods invoked
- M_h:the number of methods inherited
- Ma:the number of methods referencing attributes of another class
- Mp:the number of methods having parameter of the type of another class
- {}:representation of set of all elements considered in the algorithm

$$CLC_{(i,j)} = \sum_{i=1}^{i=n}\left(\frac{m_h+m_a+m_p+m_j}{n*c}\right) \quad ….(4)$$

3.2 Object Level Coupling

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
$|Z|$: 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.

$$OLC_x(O_iO_j) = \sum_{Z=1}^{Z=n}\left(\ \frac{M}{Z}\right) \quad ………. (5)$$

3.3 Dynamic Level Coupling

To effectively evaluate dynamic coupling level, 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

$$D_{ca} = \frac{C_{am}}{C_t} \quad ……………… (6)$$

Where Cam is the number of classes accessing methods and Ct 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.

$$D_{kc} = \frac{Calls_{sent}}{ST_{calls}} \quad ……………… (7)$$

Where ST is total static calls in the class

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.

$$P_{ac} = \frac{SRM}{TTC} \quad ………………(8)$$

Where SRM is the sum of receiving and sending classes and TTC total number of classes in the program
Representing the three compounds as contribution to the overall DCL we can say that

$$DLC = (D_{ca} + D_{kc} + P_{ac}) \quad /TC \quad ………(9)$$

3.4 Static Level Coupling

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

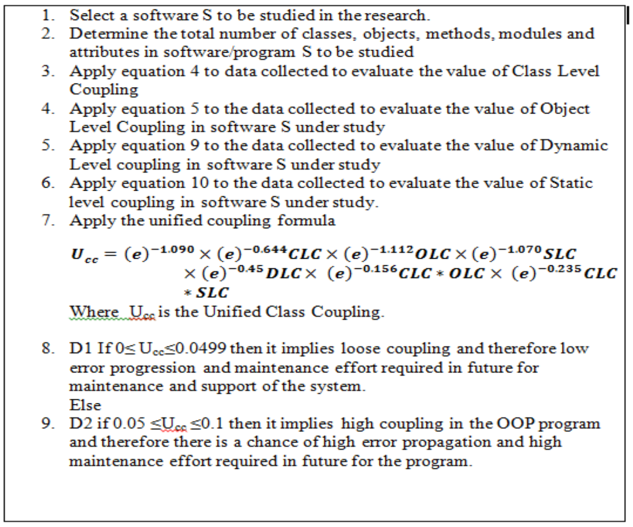
$$SLC = \sum_{c=1}^{c=n} \left(\frac{I_m}{T_m}\right) / TTC \text{ ……………………. (10)}$$

Where I_m is is the set of all interacting modules in the class under study

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

TTC is total number of classes to be studied.

3.5 Algorithm for Coupling Measurement



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 4 to data collected to evaluate the value of Class Level Coupling
4. Apply equation 5 to the data collected to evaluate the value of Object Level Coupling in software S under study
5. Apply equation 9 to the data collected to evaluate the value of Dynamic Level coupling in software S under study
6. Apply equation 10 to the data collected to evaluate the value of Static level coupling in software S under study.
7. Apply the unified coupling formula

   $$U_{cc} = (e)^{-1.090} \times (e)^{-0.644} CLC \times (e)^{-1.112} OLC \times (e)^{-1.070} SLC \times (e)^{-0.45} DLC \times (e)^{-0.156} CLC * OLC \times (e)^{-0.235} CLC * SLC$$

   Where U_{cc} is the Unified Class Coupling.

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.

Fig.1 Unified Algorithm

IV. Unified Model Assessment.

The unified model developed in this papers was subjceted to evaluation of four software programs developed and coupling values of individual components obtained and recorded in a table 2 as shown below

Table 2:Component Values

Software	CLC	OLC	DLC	SLC
OTIS	0.1111	0.261	0.243	0.400
FMA	0.125	0.629	0.292	0.455
BSA	0.167	0.794	0.458	0.7
T4S	0.167	0.618	0.567	0.533

In the evaluation process the above values were subjected to evaluation model function and table3 as shown below was created. The table indicated the values of coupling as low or high.

Table 3: Evaluation Table

APP	U_{CC}	COUPLING LEVEL
OTIS	0.000103	Low
FMA	0.001329	Low
BSA	0.02159	Low
T4S	0.0715	High

From table 3 in the evaluation table the software OTIS, fma, bsa were found to be lowly coupled while T4S was found to be highly coupled during assessment .The coupling classification was done on the basis of high for those that are 0.05 to 0.1 and lowly coupled for those that were found to be 0.049 and below.

V. Discussion

From table1 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 object oriented 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 and 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:

(i) The list of available methods is obtained

(ii) The methods are arranged in order from most specific to least specific

(iii) 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.

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. 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 object oriented programing 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. 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. 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 [1] .

Following the description some of the possible approaches for finding functionality groups of public methods are:

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

Unified coupling model borrows the concept of generalization in object oriented software systems.

In object oriented 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. 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 3

VI. Conclusion

In this paper a model for measuring coupling in object oriented software systems has been developed. The model was constructed from data analyzed in various software systems. The paper has demonstrated how to evaluate individual components of the model using varied formulas. As presented in the algorithm software systems with values less than or equal to 0.049 represents low coupling while values between 0.05 to 0.01 represents high coupling in software systems. An algorithm to assess the overall coupling has been presented. In this paper we request further research on determining effect of coupling in overall software maintenance and debugging.

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

Calvins Otieno is a PhD student in the department of computing in Jomo Kenyatta University. He has a first degree in Computer technology, Masters in Software Engineering and. 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. He has a PhD in Computer science from university of Ulcer, MSc Information Systems and BSc Mathematics and Computer Science from JKUAT.

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).