American Journal of Software Engineering and Applications
2013; 2(4): 105-110
Published online August 20, 2013 (http://www.sciencepublishinggroup.com/j/ajsea)
doi: 10.11648/j.ajsea.20130204.11

A cohesion measure for C in the context of an AOP paradigm

Zeba Khanam, S. A. M Rizvi

Jamia Millia Islamia, New Delhi

Email address:
zebs_khan@yahoo.co.in(Z. Khanam)

To cite this article:
Zeba Khanam, S. A. M Rizvi. A Cohesion Measure for C in the Context of an AOP Paradigm. *American Journal of Software Engineering and Applications*. Vol. 2, No. 4, 2013, pp. 105-110. doi: 10.11648/j.ajsea.20130204.11

Abstract: Cohesion measures the relative functional strength of a module and impacts the internal attribute of a function such as modularity. Modularity has become an accepted approach in every engineering discipline. The concept of modular design has considerably reduced the complexity of software design. It represents the strength of bond between the internal elements of the modules. To achieve effective modularity, design concepts like functional independence are considered to be very important. Aspect-oriented software development (AOSD) has emerged over the last decade as a paradigm for separation of concerns, which aims to increase the modularity. Therefore the presence of aspects affects the cohesiveness of a module. Like any new technology, aspect-oriented programming (AOP) was introduced to solve problems related to object-orientation (OO), and more in particular Java .It was noticed that AOP's ideas were not necessarily tied to OO (and Java) but also to less modular paradigm like imperative programming. Moreover, several metrics have been proposed to assess aspect-oriented systems quality attributes in an object oriented context. However, not much work has been done to assess the impact of AOP on imperative style of programming (also called procedural paradigm, such as C language). Therefore, metrics are required to measure quality attributes for AOP used with imperative programming. Cohesion is considered an important software quality attribute. In this context, this paper presents an approach for measuring cohesion based on dependence analysis using control flow graphs (CFG).

Keywords: Cohesion Measures, Procedural Paradigm, Software Metrics, Aspect Oriented Programming

1. Introduction

Module cohesion is a property of a module that represents unity of purpose. It describes the degree to which elements of a module are associated with each other. Aspect-oriented (AO) software development is a paradigm that provides new abstractions and mechanisms to support separation of concerns and the modularization of crosscutting concerns through the software development [Figueiredo etal 2005].Though there have been a number of researches on the evaluation of this design technique and it has been claimed that applying an AOSD method will eventually lead to quality software in the field of object oriented programming, however efficient evaluation in a quantitative manner about the role of AOSD in the area of procedural programming is still ignored. Moreover, phenomenon like scattering and tangling, the usual indicators for crosscutting concerns, equally arises in less modular paradigms like imperative programming. Therefore, in order to establish the significance of AOSD in improving the software attributes: maintainability, reusability and reliability of systems developed using aspect oriented techniques, software measures are required. Software engineers have assumed that the most impacted property of an aspect-oriented system is separation of concerns. However, some recent studies (e.g. [Garcia, A. et al., 2005][Garcia, A. et al.2004]) have shown that other fundamental software engineering principles, such as low coupling and high cohesion, need to be assessed in conjunction with separation of concerns issues. Cohesion describes the degree to which the actions performed within the module contribute to single behavior/function.

Module cohesion has been associated to the quality of software. Cohesion is the measure of strength of the association of elements within a module. Modules whose elements are strongly and genuinely related to each other are desired. Stevens etal. and Page-Jones claimed that cohesion is associated with effective modularity, a desirable quality of software, and has predictable effects on external software quality attributes such as modifiability, maintainability, and understandability [Yourdon and Constantine,1978][Stevens,etal 1974]. Booch has defined modularity as the property of a system whose modules are cohesive and loosely-coupled .Fenton stated that modularity is the internal quality attribute of the software system [Melton, 2007][Fenton,1994]. Karstu indicated that there appears to be a correlation between module cohesion and number of changes made to a module [Karstu, 1994] such that highly cohesive modules are less likely to need change. Though a number of papers have addressed different measures for evaluating the cohesion in procedural software[BiemanandOtt.1994][Kang &Bieman,1996] and object oriented software [Chidamber &Kemerer,1994][Briand,1998][Kiczales,1997][Chae.etal.2 000] but not much work is done when it comes to assessing the software components developed using aspect oriented programming.

In order to study the impact of aspect-oriented software development (AOSD) on evolution, one has to study its impact on software characteristics such as evolvability, maintainability, understandability, and quality. This paper addresses a measure for module cohesion for procedural software modified or refactored with Aspect oriented design and implementation .For this purpose we have used C language with AspectC as the AOP language for the quality of implementations.

The assessment of relevant attributes of aspect-oriented design and implementation is a prerequisite for achieving high-quality AO software, and that exploiting those attributes will open up a broader design evaluation, which is essential to allow the AO software engineers reason about and make a proper trade-off analysis between different solution alternatives.

The rest of the paper is organized as follows. Section 2 briefly describes the related work. Section 3 introduces Aspect oriented programming and Aspect. Section 4 depicts the application of AOP in procedural software. Section 5 presents member dependency in aspect oriented scenario. Section 6 defines a cohesion measure suite based on the dependence criteria. Section 6

2. Related Work

There are number of research work dedicated to measure and analyze the complexity of software systems [Chidamber, 1994] [Buse, 2008] [McCabe, 1976] [Halstead, 1979]. Several metrics have been proposed in the literature in order to assess quality attributes (complexity, coupling, cohesion, etc.)Software metrics measures the complexity of software systems for software cost estimation, software development control, software assurance, software testing, and software maintenance. Several software metrics exist based on different categories [Meyer, 2009]:

Size-related software metrics: NCLOC, Memory footprint, Number of classes / headers, Number of methods, Number of attributes, Size of compiled code, etc.

Quality-related software metrics: Cyclomatic complexity, Number of states, Number of bugs in LOC, Coupling metrics, Inheritance metrics, etc.

Process-related software metrics: failed builds, defect per hour, requirement changes, programming time, number of patches after release, etc.

There are currently more than 200 metrics with many different purposes [Meyer, 2009], but currently, the existing procedural metrics [Henryan&Kafura, 1981][McCabe,1976] are only applicable on the procedural software not aspect oriented code therefore if the aspect oriented constructs are intercepting the source files it is necessary to evaluate their impact because they tend to affect the cohesion and coupling between the modules and the introduced advice. McCabe measures the number of linearly independent paths through a program's source code.[McCabe 1976] proposed complexity measures based on the number of local information flows entering and exiting in each module. Presently, a number of papers have addressed the metrics related to aspect-oriented programs quality [Zhao&Xu,2004][Kang&Bieman,1996][Gélinas,2006][Ceccato&Tonella,2004][Anna.etal.2003].One of the first approaches in the field of cohesion measurement for AOP was given by Zhao. It is based on a dependency model for aspect-oriented software that consists of a group of dependency graphs. According to Zhao and Xu's approach, cohesion is defined as the degree of relatedness between attributes and modules. Zhao and Xu present, in fact, two ways for measuring aspect cohesion based on inter-attributes (γa), inter-modules (γm) and module-attribute (γma) dependencies. Further, this approach was modified by [Gélinas etal, 2006].They analyzed that the approach was complicated and the cohesion(x) computation was based on some arbitrary constants $\beta 1$, $\beta 2$, and $\beta 3$.

(Where $x = \beta 1 * \gamma a + \beta 2 * \gamma m + \beta 3 * \gamma ma$, k the number of attributes and n the number of modules in aspect A). [Gelinas etal 2006] deviced a measure for cohesion computation based on data-module and module-module connection criteria. Therefore, the Aspect cohesion (ACoh) was computed as:

ACoh represents the relative number of connected modules: ACoh(Aspecti) = NC(Aspecti) / NM(Aspecti) Є [0,1].

Where NM(Aspecti) is the total number of modules pairs in an aspect and NC(Aspecti) is the number of connections between modules. The target AOP language was AspectJ.

In [Anna.etal, 2003] a method for the computation of LCOM was derived from the well-known LCOM (Lack of Cohesion in Methods) metric developed by [Chidamber&Kemerer,1994].A more synonymous extension of C&K metric suite [Chidamber&Kemerer,1994] has been made in [Cecatto&Tonella,2004] but it is again a measure for object oriented software designed using AspectJ as the aspect oriented language. Therefore all the measures are basically devised for aspect oriented systems developed in an object oriented environment. We are inspired by some approaches proposed for cohesion measurement [Briand, 1998][Garcia A.etal,2004][Zhao&Xu,2004][Gélinas,etal 2006]. As none of the existing metrics tend to target the AO code in AspeCt C we have devised a measure for computing cohesion in AOP used with C software.

3. Aspect-Oriented Programming and Aspect-Oriented C

In this paper we use C and AspectC as the aspect oriented language to show the basic ideas of coupling measurement in AO systems. AspectC is an aspect-oriented extension to C by adding some new concepts and associated constructs. The current ACC language design adapts the ideas of AOP introduced by Kiczales [Kickzales1997] to the C programming language. These concepts and associated constructs are called join points, pointcut, advice, intype and introduce declaration, and aspect. The AspeCt-oriented C compiler processes the advice declaration from the aspect file and the core program from the core file and generates C sources that contain information from both files. This step is referred to as aspect compilation. That is the advice specified in the aspect file is woven into the core to result in a program that reflects both programs' intends.

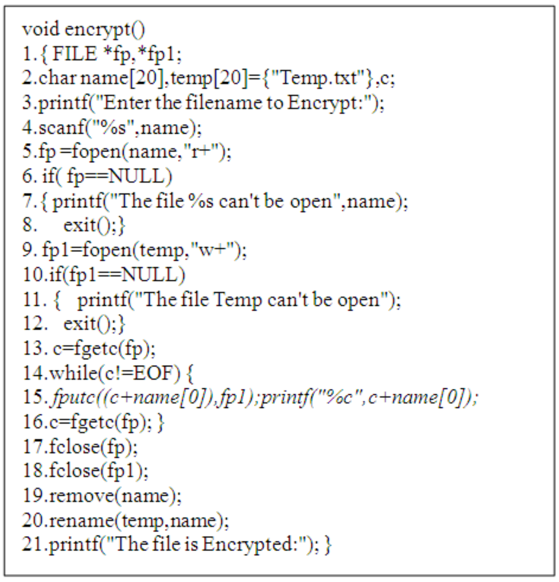
The major construct of AspectC is the advice, which is just like function but are executed when a join point is matched by a pointcut defined inside the code part of a pointcut declaration. They are different from the aspects in AspectJ, as the aspects are just like classes that encapsulate functionalities that crosscut other classes.

4. Applicability of AOP in Procedural Software

We have used an example from an encryption program to make a reusable aspect that checks the file opening result. Below is an example to depict a simple encryption function. In order to encrypt or decrypt the file, it needs to be opened and the file check operation has to be done to ensure that the file pointer doesn’t return null.

The check is done after each call to fileopen().Hence all the operations to be carried on the file uses the above code fragment that is almost identically scattered across the whole system. This is a very important check that needs to be performed but at the same time the code unnecessarily distracts from the principal program logic. This is an example of an aspect. This therefore reduces the understandability of the code and also if the code needs updation it needs to be done at several places, which unnecessarily creates complications.

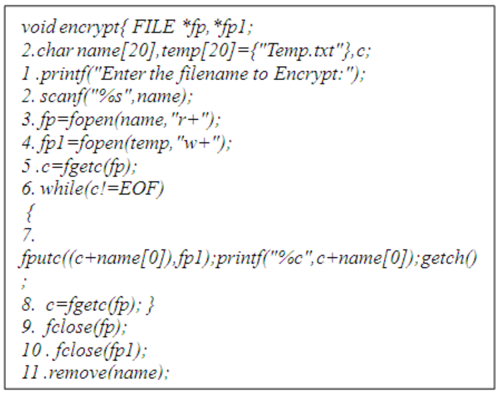
Example: It is common practice to check the return value after opening a file for any use to ensure the return value is non null. This code often looks as follows:



```
void encrypt()
{ FILE *fp,*fp1;
char name[20],temp[20]={"Temp.txt"},c;
printf("Enter the filename to Encrypt:");
scanf("%s",name);
fp =fopen(name,"r+");
if( fp==NULL)
{ printf("The file %s can't be open",name);
   exit();}
fp1=fopen(temp,"w+");
if(fp1==NULL)
{   printf("The file Temp can't be open");
 exit();}
c=fgetc(fp);
while(c!=EOF) {
fputc((c+name[0]),fp1);printf("%c",c+name[0]);
c=fgetc(fp); }
fclose(fp);
fclose(fp1);
remove(name);
rename(temp,name);
printf("The file is Encrypted:"); }
```

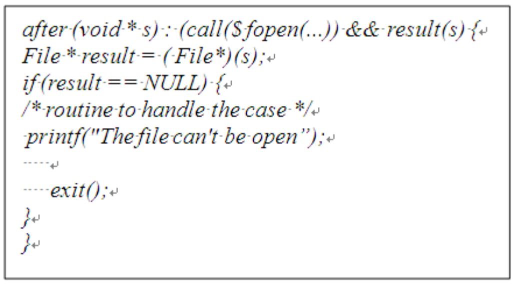
Figure 1. *Code snippet from file encryption program*

Thus a better option is to isolate the concern that would improve maintainability and would also better modularize the system. In the above example the file checking logic would be extracted into an aspect file, as follows:



```
void encrypt{ FILE *fp, *fp1;
char name[20],temp[20]={"Temp.txt"},c;
printf("Enter the filename to Encrypt:");
scanf("%s",name);
fp=fopen(name,"r+");
fp1=fopen(temp,"w+");
c=fgetc(fp);
while(c!=EOF)
{
fputc((c+name[0]),fp1);printf("%c",c+name[0]);getch()
;
 c=fgetc(fp); }
 fclose(fp);
fclose(fp1);
remove(name);
```

Figure 2. *Aspect to handle the checking logic*



```
after (void * s) : (call($ fopen(...)) && result(s) {
File * result = (File*)(s);
if (result == NULL) {
/* routine to handle the case */
printf("The file can't be open");

    exit();
}
}
```

Figure 3. *File Encryption Code after removing the file check routine*

A similar situation arises when malloc( ) and calloc( ) functions are used for memory allocation. After each call to malloc ( ) it is a common practice to check if the value returned after memory allocation is null or not null. The memory checking concern is scattered throughout the entire program hence crosscutting each function, therefore the AspectC offers a good solution by the extraction of the concern in an advice that should be invoked after each call to these functions.

5. Member Dependency in an Aspect Oriented Scenario

As illustrated in the previous section, different cohesion measures have been proposed by Zhao, Ceccato and Jean. But all these measures are specially designed for AspectJ but AspectC has different constructs and the structure of the procedural program too is different from the object oriented constructs so none of these measures can be applied directly. However, in the context of the measures defined by Zhao we have defined the member dependencies in C and AOP and subsequently the measure to quantify cohesion.

Our basic concepts will be illustrated using AspectC. AspectC introduces several new language constructs such as: join points, pointcuts, advice as well as intype declarations. Join points are well-defined points in the structure and dynamic execution of a system. Examples of join points are method calls, method executions, and field sets and reads. Pointcuts describe join points and context to expose. Advice is a method-like abstraction that defines code to be executed when a join point is reached. Pointcuts are used in the definition of an advice. Inter-type declarations define how an aspect modifies a program's static structure, namely, the members and the relationship between components. Pointcuts and advice dynamically affect program flow, and inter-type declarations statically affect a program's class hierarchy.

Since in a procedural language a program is written using functions or procedures, therefore cohesiveness is defined on a software module/function/procedure. Therefore cohesion in a function is an internal software attribute that measures the degree to which its members are bounded together. Cohesion can be a measure to identify the poorly designed function or advice. A function that has probably been assigned unrelated concerns will depict a low cohesion value. Thus such a function will be difficult to understand, to test, to reuse and to maintain but if a concern is extracted into an advice and is separated from the original function then the function exhibits higher cohesiveness and if the advice handles a single concern then it is also supposedly highly cohesive.

The cohesion measure is defined on the basis of dependence analysis. We define cohesion on the basis of inter attribute dependence. Therefore we present the dependency between the attributes defined in a function or an advice and the dependencies are depicted using Control flow Graph (CFG) of a module.

The associations (or relationships) between the processing elements of a module are defined in terms of control and data dependencies between the variables of a module. These dependencies are computed from a directed graph called Control Flow Graph (VDG). Control flow analysis is defined as:

Definition 1: The control flow graph, or simply a flow graph, of a program is a directed graph where the nodes correspond to the basic blocks of the program and the edges represent potential transfer of control between two basic blocks [Aho86, Hecht77].

Dependence Definition: Consider a directed Control Flow Graph (CFG) of a module M (i.e. a module is used for a function or an advice) GM where the nodes represent the basic blocks of the module and the edges represent the control transfer between the 2 blocks. A basic block is a group of statements such that no transfer occurs into a group except to the first statement in that group, and once the first statement is executed, all statements in the group are executed sequentially [Hecht77].If there are n attributes a1, a2,…an in the module, then any attribute a1 € GM is said to be related to ai € GM if they both lie on the same edge of the GM and is denoted as a1 → ai.

Therefore, the relatedness or the dependence of an attribute ai to the other attributes of the module is computed at every edge of the CFG GM.

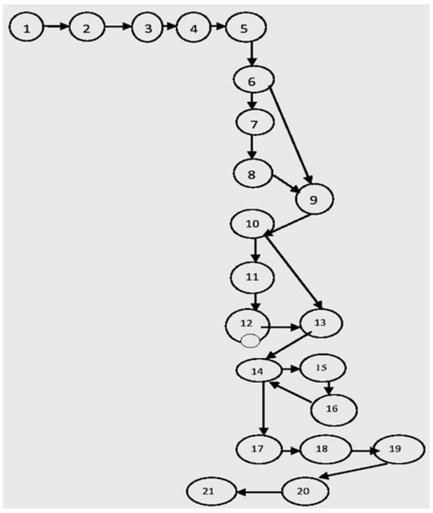


Figure 4. *Contror Flow Graph For Encrypt().*

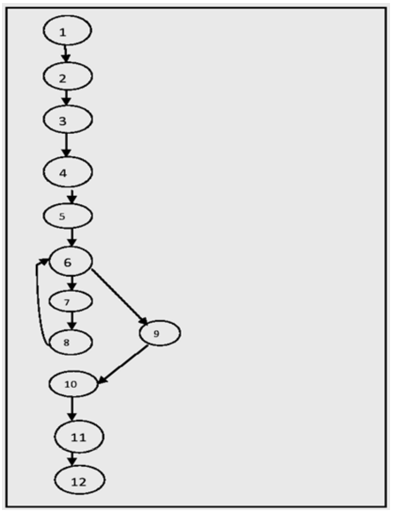


Figure 5. *CFG for Encrypt after Refactoring*

6. A Cohesion Measure

The cohesion is about tightness between attributes in a module. Based on the introduced dependence criteria, we define the cohesion of a module (advice/function) by the degree of relatedness of its attributes or data. We are inspired by some approaches proposed for aspects and class cohesion measurement [Zhao,2003][Zhao&Zu,2004][Gélinas, etal 2006]. To compute the cohesion for a module, we define it as follows:

Definition: For each attribute ai of module M, a set $R_M(a)$ contains the attributes on which 'a' depends or is related to by the dependence criteria defined above. Thus

$R_M(a)=\{ai|\ a\rightarrow ai, a\neq ai\}$ where i=1,2…..k such that k is the number of attributes present in the module M. We define the degree of relatedness of 'a' to the other attributes of the module 'M' on the graph 'GM' with 'E' edges as follows:

$$DR\ (a)=1/e\sum_{i=1}^{e} |R_M(a)|/(k-1)$$

k represents the number of attributes:- e represents the blocks that are not declarative statements.
Thus we define the cohesion measure as:

$$Ca(M) =\{\ 0\ k=0\ 1\ k=11/k \sum kj=1\ DR(aj)\ k>1\ \}$$

The cohesion is measured on the basis of attributes present. If the attribute is zero then the inter attribute cohesion is 0 if there is a single attribute then the cohesion is 1 and one attribute itself is tight.

Example 1: The degree of cohesion is computed through the CFGs available (Fig.4 and Fig.5). For the function encrypt, before refactoring Ca (encrypt) is .705(.705/5=.141) based on the above definition of cohesion. After refactoring the cohesion measure is Ca (encrypt) =.179 and can be observed that it has increased. As we compute the aspect cohesion for the advice we may notice that there is a single attribute in the function so therefore the inter attribute cohesion is 1 for the advice.

6. Conclusion

The application of a particular metrics to software is dependant upon the system properties that are to be assessed. Modular software has several advantages such as maintainability, manageability, and comprehensibility. As described many researchers, five attributes are closely related to modularity in software systems which are coupling / dependency, complexity, cohesion, and information hiding.

Thus, cohesion is an important internal attribute of a software that affects the modularity of a software and hence the maintainability of software. This paper proposes a cohesion measure for assessing the cohesiveness of a module (function/advice) in C in the context of AOP (AspectC) environment. The proposed cohesion measure is basically defined for procedural software using an aspect oriented environment. Cohesiveness Measure Ca(M) of a module (advice/function) is defined on the basis of its inter attribute dependence, as attributes form the basic building blocks of a module and their correlation forms the functionality of the module and is the most significant aspect for establishing the tightness of a module. We had also discussed the criteria of attribute dependence and have depicted an AOP scenario with an example to compute its cohesion measure.

Most of the metrics suites for Aspect oriented software are defined in the context of object oriented programming. Therefore, we believe that our approach can be a good measure to assess the cohesion of a procedural module in an AOSD context. In future, we intend to perform more empirical studies in order to establish the tradeoff between advantages and disadvantages obtained by using the AOP approach in terms of other software attributes.

References

[1] E. Figueiredo, A. Garcia, C. Sant'Anna, U. Kulesza, and C. Lucena, Assessing Aspect-Oriented Artifacts: Towards a Tool-Supported Quantitative Method, Wkshp. on Quantitative Approaches in OO Software Engineering, 2005.

[2] S.R. Chidamber and C.F. Kemerer, A Metrics suite for object Oriented Design, IEEE Transactions on Software Engineering, Vol. 20, No. 6, pp. 476-493, June 1994.

[3] J.Bieman and L. Ott.Measuring Functional Cohesion. IEEE Transactions of Software Engineering.Vol.22,No.10,August 1994.

[4] L.C. Briand, J. Daly and J. Wusr, A unified framework for cohesion measurement in object-oriented systems, Empirical Software Engineering, Vol.3, No.1, pp. 67-117, 1998.

[5] G.Kiczales, J.Lamping, A.Mendhekar,C.Maeda,C.Videara Lopes,J.M. Loingtier and J.Irwin,Aspect Oirented Programming. In ECOOP,1997.

[6] H.S. Chae, Y. R. Kwon and D H. Bae, A cohesion measure for object-oriented classes, Software Practice and Experience, No. 30, pp. 1405-1431, 2000.

[7] Garcia, A. et al.: Modularizing Design Patterns with Aspects: A Quantitative Study. In Proc. of the AOSD'05, Chicago, USA, (2005), pp. 3-14.

[8] Garcia, A. et al.: Separation of Concerns in Multi-Agent Systems: An Empirical Study. In Software Engineering for Multi-Agent Systems II, Springer, LNCS 2940, (2004).

[9] J. Zhao, Coupling Measurement in Aspect-Oriented Systems, Technical-Report SE-142-6, Information Processing Society of Japan (IPSJ), July 2003.

[10] J. Zhao and B. Xu, Measuring Aspect Cohesion, Proceeding of International Conference on Fundamental Approaches to Software Engineering (FASE'2004), LNCS 2984, pp.54-68, Springer-Verlag, Barcelona, Spain, March 29-31, 2004.

[11] B.Kang and Bieman, Design Level Cohesion Measures:Derivation,Comparisons and Applications,Computer Science Technical Report CS-96-103,Colorado State University,1996.

[12] J.F. Gélinas, L. Badri and M. Badri, A Cohesion Measure For Aspects, in Journal of Object Technology, vol. 5, no. 7, September - October 2006, pp. 97 – 114 http://www.jot.fm/issues/issue_2006_09/article5.

[13] Henry, S., Kafura, Software Structure Metrics Based on Information Flow D. IEEE Transactions on Software Engineering Volume SE-7, Issue 5, Sept. 1981 Page(s): 510 - 518

[14] McCabe,T.,A Software Complexity Measure,IEEE Transactions on Software Engineering,Vol 2,Issue 4,pp 308-320,1976.

[15] Meyer B., Oriol M., & Schoeller B. (2009), "Software engineering: lecture 17-18: estimation techniques and

[16] software metrics", Chair of Software Engineering Website, available: http://se.inf.ethz.ch/teaching/2008-S/se0204/slides/15-Estimation-and-metrics-1-6x.pdf , accessed: 18 January 2009.

[17] N. E. Fenton.(1994) "Software Measurement: A necessary scientific basis", IEEE Trans. Software Eng., vol. 20,no. 3, March 1994, pp. 199-206.

[18] Mariano Ceccato and Paolo Tonella,(2004) " Measuring the Effects of Software Aspectization", In Cd-rom Proceedings of the 1st Workshop on Aspect Reverse Engineering (WARE 2004). November, 2004. Delft, The Netherlands.

[19] C. Sant‟Anna, A. Garcia, C. Chavez, A. von Staa, and C. Lucena. On the reuse and maintenance of aspect oriented software: An evaluation framework. In 17o. Simpsio Brasileiro de Engenharia de Software, pages 19–34,2003.

[20] Yourdon, E. and Constantine, L. L., Structured Design, Yourdon Press, 1978.

[21] Stevens, W. P., Myers, G. J. and Constantine, L. L., "Structured Design," IBM Systems Journal, Vol. 13, No. 2, May 1974.

[22] Karstu , S., An Examination of the Behavior of Slice Base Cohesion Measures, Master's Thesis, Michigan Technological University, Department of Computer Science, August 1994.