International Journal of Modern Engineering Research (IJMER)
www.ijmer.com Vol.1, Issue1, pp-113-119 ISSN: 2249-6645

FAULT PREDICTION IN OBJECT-ORIENTED SYSTEMS BASED ON C^3 (CONCEPTUAL COHESION OF CLASSES)

PRAKASA RAO DASARI
2/2 M.TECH CSE, DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
ADITYA INSTITUTE OF TECHNOLOGY AND MANAGEMENT, TEKKALI
ANDHRA PRADESH, INDIA

VASANTHAKUMARI G
ASSOC.PROFESSOR
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
ADITYA INSTITUTE OF TECHNOLOGY AND MANAGEMENT, TEKKALI
ANDHRA PRADESH, INDIA

Abstract—**To remain competitive in the dynamic world of software development, organizations must optimize the usage of their limited resources to deliver quality products on time and within budget. This requires prevention of fault introduction and quick discovery and repair of residual faults. In this paper a new approach for predicting and classification of faults in object-oriented software systems is introduced. In particular, cohesion is a desirable property of software as it positively impacts understanding, reuse, and maintenance. Currently proposed measures for cohesion in Object-Oriented (OO) software reflect particular interpretations of cohesion and capture different aspects of it. Existing approaches are largely based on using the structural information from the source code, such as attribute references, in methods to measure cohesion. This paper proposes a new measure for the cohesion of classes in OO software systems based on the analysis of the unstructured information embedded in the source code, such as comments and identifiers.**

Index Terms— **Fault Prediction; Object Oriented Software; Conceptual Cohesion of Classes(C^3)..**

I. INTRODUCTION

Software reliability can be defined as the probability of failure-free operation of a computer program executing in a specified environment for a specified time [1]. It is often considered a software quality factor that can aid in predicting the overall quality of a software system using standard predictive models. Predictive models of software faults use historical and current development data to make predictions about faultiness of software subsystems/modules. One of the goals of the OO analysis and design is to create a system where classes have high cohesion and there is low coupling among them. These class properties facilitate comprehension, testing, reusability, maintainability, etc. Software cohesion can be defined as a measure of the degree to which elements of a module belong together [2]. Cohesion is also regarded from a conceptual point of view. In this view, a cohesive module is a crisp abstraction of a concept or feature from the problem domain, usually described in the requirements or specifications. Although software faults have been widely studied in both procedural and object-oriented programs, there are still many aspects of faults that remain unclear. This is true especially for object-oriented software systems, in which inheritance and polymorphism can cause a number of anomalies and fault types [3]. Unfortunately, existing techniques used to predict faults in procedural software are not generally applicable in object-oriented systems.

Proposals of measures and metrics for cohesion abound in the literature as software cohesion metrics proved to be useful in different tasks [4], including the assessment of design quality [5], [6], productivity, design, and reuse effort, prediction of software quality, fault prediction , modularization of software, and identification of reusable of components [7]. Most approaches to cohesion measurement have automation as one of their goals as it is impractical to manually measure the cohesion of classes in large systems. We propose a new measure for class cohesion, named the Conceptual Cohesion of Classes (C3), which captures the conceptual aspects of class cohesion, as it measures how strongly the methods of a class relate to each other conceptually.

A. *Class Cohesion*

The components of a class are the instance variables and methods defined in the class plus those that are inherited. A method and an instance variable are related by the way that an instance variable is used by the method. Two methods are related (connected) through inst ante variable(s) if both methods use the instance variable(s). Class cohesion is defined in terms of the relative number of connected methods in the class.

B. *Inheritance and Cohesion*

A subclass inherits methods and instance variables from its super class. We have several options for evaluating cohesion of a subclass. We can (1) include all inherited components in the subclass in our evaluation, (2) include only methods and inst ante variables defined in the subclass, or (3) include inherited instance variables but not inherited methods. The class cohesion measures that we develop can be applied using any one of these options.

C. *Measuring Object Oriented Reuse*

We focus on private reuse within one software system [8]. We evaluate reuse from the server perspective, since this is the best orientation for evaluating reusability [9]. We are interested in two different forms of class reuse, reuse via instantiation and reuse via inheritance.

A class is reused by being instantiated in other classes or by being inherited to them. Instantiation reuse of a class is measured as the number of classes where the class is instantiated. Inheritance reuse of a class is the number of classes which inherit the class, i.e., the number of descendants (both direct and indirect descendants).

II. Related Work

Software developers aim for systems with high cohesion and low coupling. The value of these goals has not been validated empirically [10]. Rather, they have been justified on the basis of intuition. The amount of reuse the number of times that a component is reused is an indicator of reusability. Of course, other factors such as the usefulness of a component are also components of reusability.

Cohesion refers to the “relatedness” of module components. A highly cohesive component is one with one basic function. It should be difficult to split a cohesive component. Cohesion can be classified using an ordinal scale that ranges from the least desirable category coincidental cohesion to the most desirable functional cohesion [11]. To apply this cohesion model to classes in object-oriented software, we need to add a new classification, data cohesion [8].

Bieman and Ott developed a set of functional cohesion measures based on program slices [12]. These measures apply only to individual functions; their application to entire classes is not obvious. Chidamber and Kemerer developed a Lack of Cohesion in Methods (LCOM) measure for object-oriented software [13]. LCOM is effective at identifying the most non-cohesive classes, but it is not effective at distinguishing between partially cohesive classes. LCOM indicates lack of cohesion only when, compared pair wise, fewer than half of the paired methods use the same instance variables.

Recently, other structural cohesion metrics have been proposed, trying to improve existing metrics by considering the effects of dependent instance variables whose values are computed from other instance variables in the class [14], [15], [16]. Other recent approaches have addressed class cohesion by considering the relationships between the attributes and methods of a class based on dependence analysis [17]. Although different from each other, all of these structural metrics capture the same aspects of cohesion, which relate to the data flow between the methods of a class.
Even though these metrics were not specifically designed for the measurement of cohesion in OO software, they could be extended to measure cohesion in OO systems.

III. An Information Retrieval Approach to Class Cohesion Measurement

OO analysis and design methods decompose the problem addressed by the software system development into classes in an attempt to control complexity. High cohesion for classes and low coupling among classes are design principles aimed at reducing the system complexity. The most desirable type of cohesion for a class is model cohesion [18] such that the class implements a single semantically meaningful concept. This is the type of cohesion that we are trying to measure in our approach.

The source code of a software system contains unstructured and (semi)structured data. The structured data is destined primarily for the parsers, while the unstructured information (that is, the comments and identifiers) is destined primarily to the human reader. Our approach is based on the premise that the unstructured information embedded in the source code reflects, to a reasonable degree, the concepts of the problem and solution domains of the software, as well as the computational logic of the source code. This information captures the domain semantics of the software and adds a new layer of semantic information to the source code, in addition to the programming language semantics.

A. An example of measuring C3

To better understand the C3 metric, consider a class c ∈ C with five methods m1, m2, m3, m4, m5. The conceptual similarities between the methods in the class are shown in Table 1. For the computation of ACSM we consider all pairs of different methods, thus ACSM(c) = 0.5. Since the value is positive, C3(c) = ACSM(c) = 0.5. This particular value for C3 does not indicate high cohesion for class c nor a low one, but the CSM values from Table 1 show that m1 and m3, m2 and m4, m2 and m5, and m4 and m5 are closely related respectively (i.e., the CSM between each pair is larger than C3). As one can see in this example, CSM is not a transitive measure. Since C3 is an average measure, we could have situations when some pairs of methods are highly related and other are not and the average is around 0.5. With that in mind, we refine the C3 to measure the influence of the difference between the highly related and unrelated pairs of methods on the cohesion of the class.

Table 1. Conceptual similarities between the methods in class c. ACSM(c) = 0.5.

	m1	m2	m3	m4	m5
m1	1	0.21	0.72	0.33	0.42
m2		1	0.28	0.91	0.66
m3			1	0.37	0.27
m4				1	0.89
m5					1

B. An example of measuring LCSM

Consider the same class c described in section 3.3 with C3(c) = 0.5. For each method of the class c, we compute Mi based on definition 4: M1 = {m3}, M2 = {m4, m5}, M3 = {m1}, M4 = {m2, m5}, M5 = {m2, m4}. Table 1 shows us the intersection among all pairs of sets Mi ∩ Mj in class c. Based on the intersection P = {(M1, M2); (M1, M3); (M1, M4); (M1, M5); (M2, M3); (M3, M4); (M3, M5)} and |P| = 7. Q = {(M2, M4); (M2, M5); (M4, M5)} and |Q| = 3. Thus, LCSM(c) = 7-3 = 4.

Table 2. Intersection results for method sets

	M1	M2	M3	M4	M5
M1		Ø	Ø	Ø	Ø
M2			Ø	m5	m4
M3				Ø	Ø
M4					m4
M5					

The two results combined indicate a lower value for the cohesion of class c from the example. In another situation, class c’ could have had more highly related methods than in this case (i.e., four pairs) and less unrelated method pairs with the same C3(c’) value (i.e., 0.5). Assume Table 2 would indicate 6 pairs of method sets with non empty intersection and only 4 with an empty intersection. The LCSM(c’) in that case would be 0. The combined measures will indicate that c’ is more cohesive than c.

IV. SYSTEM ARCHITECTURE

The measuring methodology for the proposed cohesion metrics is described in Figure 1. The following steps are necessary to compute the C3 and LCSM metrics:

- Preprocessing and parsing of the source code to produce a text corpus. Comments and identifiers from each method are extracted and processed. A document in the corpus is created for each method in every class.
- An IR method is used to index the corpus and create an equivalent semantic space.

- Based on the IR indexing conceptual similarities are computed between each pair of methods.

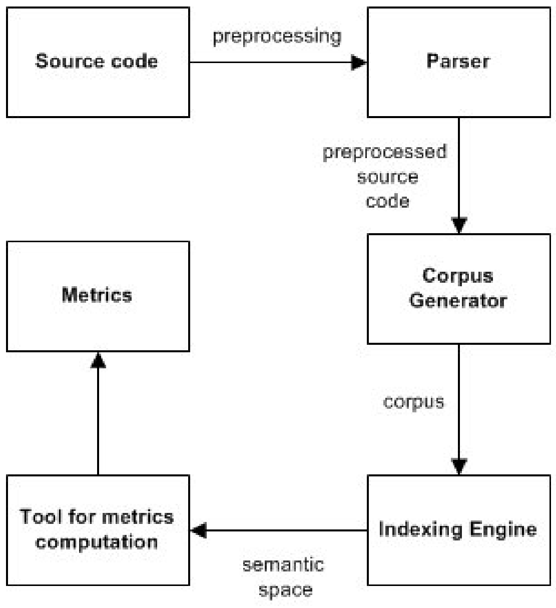


Fig 1. Measuring methodology and tools

We implemented a tool to compute C3 and LCSM for C++ software projects in MS Visual Studio .NET, based on the above methodology. Our source code parser component is based on the “Visual C++ Object Extensibility Model”. Using project information retrieved from Visual Studio .NET, the tool retrieves parts of source code that are used to produce a corpus. The extracted comments and identifier are processed by elimination of stop words and splitting identifiers that follow predefined coding standards. The corpus is indexed by the indexing engine, which is an implementation of LSI. We use the cosine between vectors in the LSI space to compute conceptual relations.

V. IMPLEMENTATION

We developed a tool IR-based Conceptual Cohesion Class Measurement, which supports this methodology and automatically computes C3 for any class in a given software system. The following steps are necessary to compute the C3 metric:

A. Corpus creation

The source code is preprocessed and parsed to produce a text corpus. Comments and identifiers from each method are extracted and processed. A document in the corpus is created for each method in every class.

B. Corpus indexing

LSI is used to index the corpus and create an equivalent semantic space. Computing conceptual similarities. Conceptual similarities are computed between each pair of methods.

- Based on the conceptual similarity measures, C3 and LCSM are computed for each class.

C. Computing C3

Based on the conceptual similarity measures, C3 is computed for each class (definitions are presented in the next section). IRC3Mis implemented as an MS Visual Studio .NET addin and computes the C3 metric for C++ software projects in Visual Studio based on the above methodology. Our source code parser component is based on the Visual C++ Object Extensibility Model. Using project information retrieved from Visual Studio .NET, the tool retrieves parts of the source code that are used to produce a corpus. For software projects that are developed outside the .NET environment, that is, Mozilla from our case study, we use external parsers and a set of our own utilities to construct the corpus. The extracted comments and identifiers are processed in the elimination of stop words and splitting identifiers that follow predefined coding standards. We use the cosine between vectors in the LSI space to compute conceptual relations. A Java version of the tool is being developed as an Eclipse plug-in.

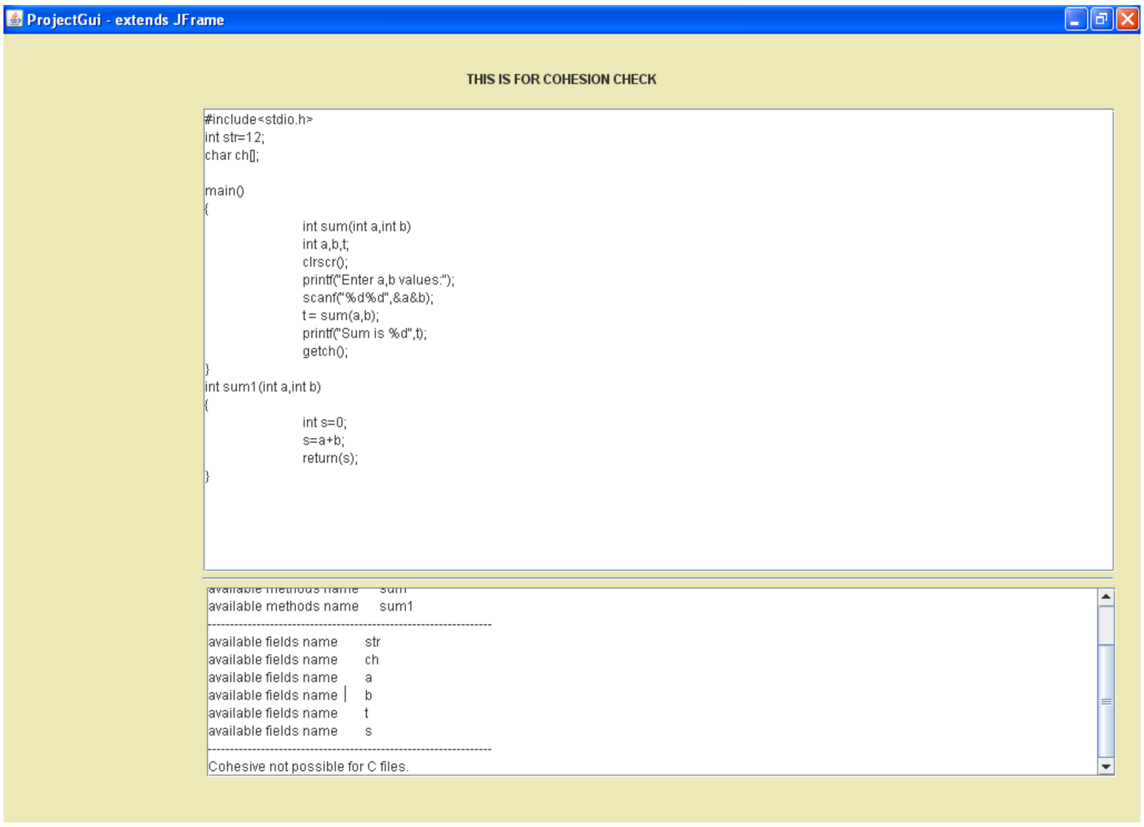
VI. RESULTS

First, we performed the univariate logistic regression The R2 coefficient is defined as the proportion of the total variation in the dependant variable y (the fault proneness of a class) that is explained by the regression model. The bigger the value of R2, the larger the portion of the total variance in y that is explained by the regression model and the better the dependent variable y is explained by the explanatory variables.

In order to evaluate logistic regression models based on the studied metrics and their combinations, we utilize the following quantitative characteristics: precision, correctness, and completeness. We use these measures to be consistent with previously published results [19, 20]. Note that these characteristics of the results are somewhat different from the precision and recall measures used in IR.

Precision here is used to evaluate how well the model classifies faulty and nonfaulty classes. For example, C3 used as a separate explanatory variable in the univariate logistic model classified 1,267 (667 as nonfaulty + 600 as faulty) classes correctly out of 2,042 classes for Mozilla, that is, a precision of 62.05 percent (see Table 4). The results of the univariate logistic regression indicate that the model based on C3 is better than any other model except that of LCOM3.

Correctness is used to show what percentage of the faulty predicted classes is really faulty (computed as the number of classes observed and predicted as faulty divided by the total number of classes predicted as faulty). In the case of the univariate logistic regression model based on C3, the correctness is 61.35 percent since it suggested 978 classes as containing faults, but, in fact, only 600 of those have faults. Fig 2,34 shows the C out put file, C++ out put file and JAVA out put file respectively.

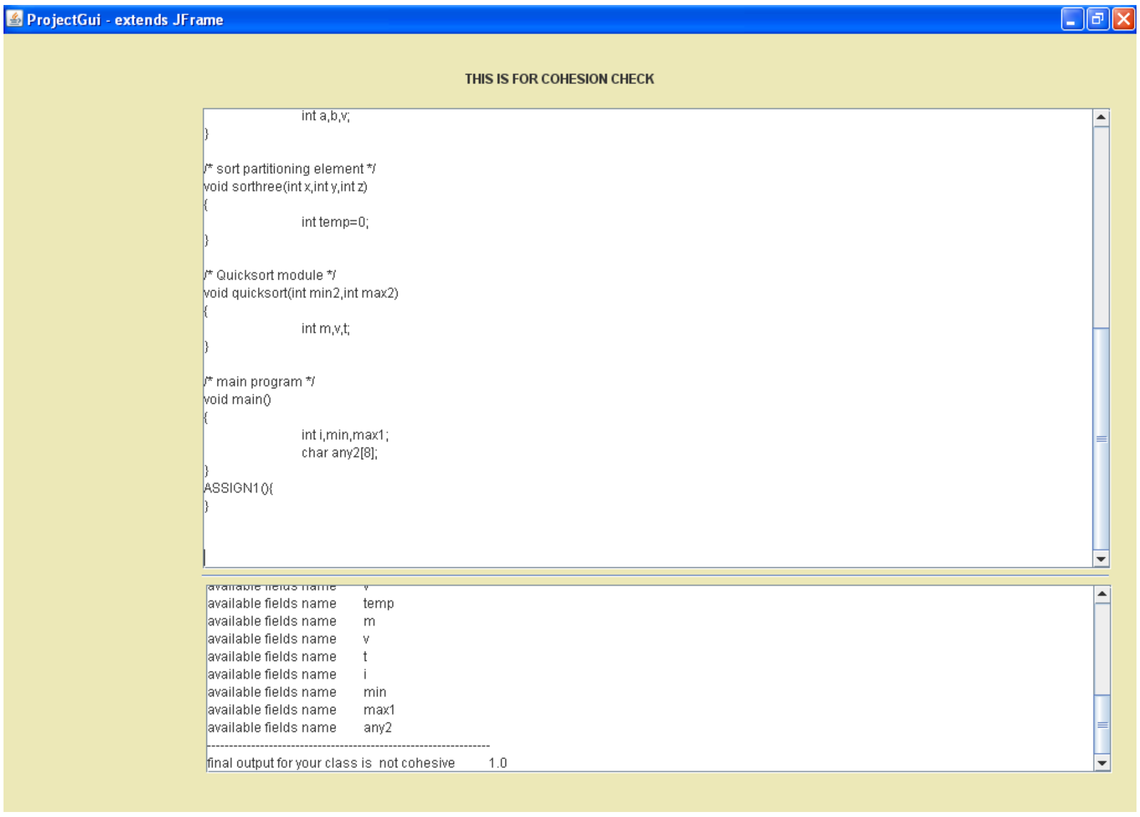


THIS IS FOR COHESION CHECK

```
#include<stdio.h>
int str=12;
char ch[];

main()
{
                int sum(int a,int b)
                int a,b,t;
                clrscr();
                printf("Enter a,b values:");
                scanf("%d%d",&a&b);
                t= sum(a,b);
                printf("Sum is %d",t);
                getch();
}
int sum1(int a,int b)
{
                int s=0;
                s=a+b;
                return(s);
}
```

```
available methods name     sum1
-------------------------------------------------------------------
available fields name      str
available fields name      ch
available fields name      a
available fields name      b
available fields name      t
available fields name      s
-------------------------------------------------------------------
Cohesive not possible for C files.
```

Fig 3 C OUTPUT File

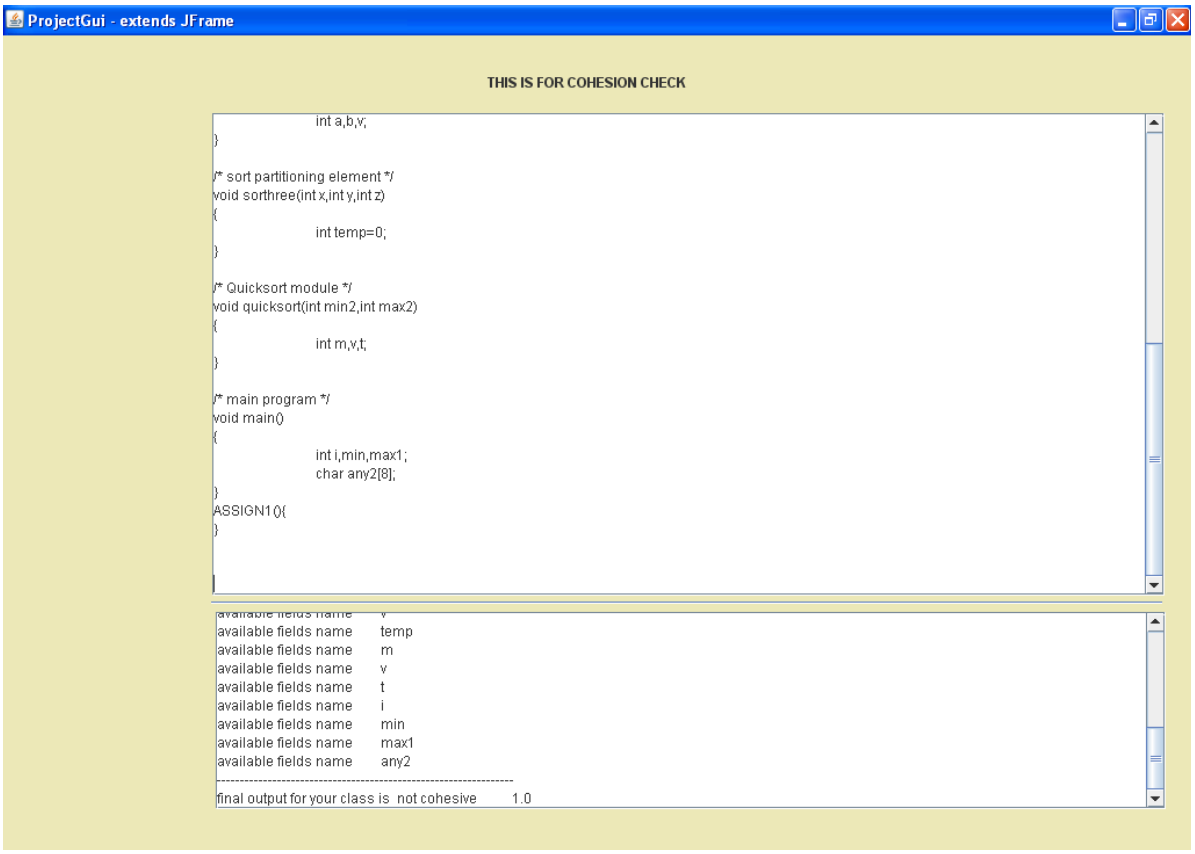


THIS IS FOR COHESION CHECK

```
                int a,b,v;
}

/* sort partitioning element */
void sorthree(int x,int y,int z)
{
                int temp=0;
}

/* Quicksort module */
void quicksort(int min2,int max2)
{
                int m,v,t;
}

/* main program */
void main()
{
                int i,min,max1;
                char any2[8];
}
ASSIGN1(){
}
```

```
available fields name      temp
available fields name      m
available fields name      v
available fields name      t
available fields name      i
available fields name      min
available fields name      max1
available fields name      any2
-------------------------------------------------------------------
final output for your class is  not cohesive       1.0
```

Fig 4 C++ OUTPUT File



```
                    int a,b,v;
}

/* sort partitioning element */
void sorthree(int x,int y,int z)
{
                    int temp=0;
}

/* Quicksort module */
void quicksort(int min2,int max2)
{
                    int m,v,t;
}

/* main program */
void main()
{
                    int i,min,max1;
                    char any2[8];
}
ASSIGN1(){
}
```

```
available fields name      v
available fields name      temp
available fields name      m
available fields name      v
available fields name      t
available fields name      i
available fields name      min
available fields name      max1
available fields name      any2
-----------------------------------------------------------------
final output for your class is  not cohesive        1.0
```

Fig 5 JAVA OUTPUT File

VII. CONCLUSIONS

Classes in object-oriented systems, written in different programming languages, contain identifiers and comments which reflect concepts from the domain of the software system. This information can be used to measure the cohesion of software. To extract this information for cohesion measurement, Latent Semantic Indexing can be used in a manner similar to measuring the coherence of natural language texts. Our results show that the classes that are heavily reused via inheritance exhibit lower cohesion. We expected to find that the most reused classes would be the most cohesive ones. Studies of additional software systems are needed to confirm these results.

Future Work

The C3 metric depends on reasonable naming conventions for identifiers and relevant comments contained in the source code. When these are missing, the only hope for measuring any aspects of cohesion rests on the structural metrics. In addition, methods such as constructors, destructors, and accessory may artificially increase or decrease the cohesion of a class. Although we did not exclude them in the results presented here, our method may be extended to exclude them from the computation of the cohesion by using approaches for identifying types of method stereotypes. C3 does not take into account polymorphism and inheritance in its current form. It only considers methods of a class that are implemented or overloaded in the class. One way in which we can extend our work to address inheritance when building a corpus is to follow the approach in, where the source code of inherited methods is included into the documents of derived classes.

References

[1]J. D. Musa, A. Iannino, and K. Okumoto, Software Reliability Measurement, Prediction, Application. the United States of America: McGraw- Hill Book Company, 1987.

[2]J. Bieman and B.-K. Kang, “Cohesion and Reuse in an Object-Oriented System,” Proc. Symp. Software Reusability, pp. 259-262, Apr. 1995.

[3]J. Offutt and R. Alexander, “A fault model for subtype inheritance and polymorphism,” in 12th International Symposium on Software Reliability Engineering, November 2001, pp. 84 – 95.

[4]D. Darcy and C. Kemerer, “OO Metrics in Practice,” IEEE Software,vol. 22, no. 6, pp. 17-19, Nov./Dec. 2005.

[5]J. Bansiya and C.G. Davis, “A Hierarchical Model for Object-Oriented Design Quality Assessment,” IEEE Trans. Software Eng., vol. 28, no. 1, pp. 4-17, Jan. 2002.

[6]L.C. Briand, J. Wu¨ st, J.W. Daly, and V.D. Porter, “Exploring the Relationship between Design Measures and Software Quality in Object-Oriented Systems,” J. System and Software, vol. 51, no. 3, pp. 245-273, May 2000.

[7]J.K. Lee, S.J. Jung, S.D. Kim, W.H. Jang, and D.H. Ham, “Component Identification Method with Coupling and Cohesion,” Proc. Eighth Asia-Pacific Software Eng. Conf., pp. 79-86, Dec. 2001.

[8]N. Fenton. Soflware Metrics - A Rigorous Approach. Chapman and Hall, London, 1991.

[9]J. Bieman. Deriving measures of software reuse in object-oriented systems. Proc, BCS-FA CS Workshop on

Formal Aspects of Measurementj pp. 79–82. Springer-Verlag, 1992.

[10] N. Fenton, S.L. Pfleeger, and R. Glass. Science and substance: a challenge to software engineers. IEEE Sofiware, 11(4):86-95, July 1994.

[11] E. Yourdon and L. Constantine. Prentice-Hall, Englewood Cliffs, Structured Design.NJ, 1979.

[12] J. Bieman and L. Ott. Measuring functional cohesion. IEEE Trans. Software Engineering, 20(8) :644–657, Aug. 1994.

[13] S. Chidamber and C. Kemerer. A metrics suite for object oriented design. IEEE Trans. Soflware Engineering, 20(6):476–493, June 1994.

[14] H.S. Chae, Y.R. Kwon, and D.H. Bae, "Improving Cohesion Metrics for Classes by Considering Dependent Instance Variables," IEEE Trans. Software Eng., vol. 30, no. 11, pp. 826-832, Nov. 2004.

[15] Y. Zhou, L. Wen, J. Wang, Y. Chen, H. Lu, and B. Xu, "DRC: A Dependence-Relationships-Based Cohesion Measure for Classes," Proc. 10th Asia-Pacific Software Eng. Conf., pp. 215-223, 2003.

[16] Y. Zhou, B. Xu, J. Zhao, and H. Yang, "ICBMC: An Improved Cohesion Measure for Classes," Proc. 18th IEEE Int'l Conf. Software Maintenance, pp. 44-53, Oct. 2002.

[17] Z. Chen, Y. Zhou, B. Xu, J. Zhao, and H. Yang, "A Novel Approach to Measuring Class Cohesion Based on Dependence Analysis," Proc. 18th IEEE Int'l Conf. Software Maintenance, pp. 377- 384, 2002.

[18] J. Eder, G. Kappel, and M. Schreft, "Coupling and Cohesion in Object-Oriented Systems," technical report, Univ. of Klagenfurt, 1994.

[19] V.R. Basili, L.C. Briand, and W.L. Melo, "A Validation of Object- Oriented Design Metrics as Quality Indicators," IEEE Trans. Software Eng., vol. 22, no. 10, pp. 751-761, Oct. 1996.

[20] T. Gyimo´thy, R. Ferenc, and I. Siket, "Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction," IEEE Trans. Software Eng., vol. 31, no. 10, pp. 897-910, Oct. 2005.