Conceptual and Semantic Measures for Cohesion in Software Maintenance

Ashutosh Mishra
Computer Engineering Department
IT-BHU, Varanasi, UP, INDIA

Vinayak Srivastava
Computer Engineering Department
IT-BHU, Varanasi, UP, INDIA

ABSTRACT
In software maintenance, cohesion plays very major role to determine the relationship among different software attributes such as class, method, function-type etc. There are many method have been used in this context such as method based on syntactically keyword count in source code. We have used the semantic value computation for the specific keyword occurs in distinct common method within the different classes for an open source code project. We have also computed the conceptual relation metric to analysis the cohesion for the method within their class. Also, there is comparison between different semantic values for the keyword of common method in this context.

Keywords
Software maintenance, cohesion, source code, semantic value, conceptual relation matrix

1. INTRODUCTION
There are many different approaches that have been done to measure cohesion in object-oriented software systems. There are many metrics such as structural metrics [3] which are able to relate object-oriented structural quality to critical reliability, maintainability, and reusability process attributes. They need appropriate measures of object-oriented structure to begin to relate structure to process. Also, [8] describes about necessity for a productive software has been culminating and object-oriented design technique which provides solution to this problem as it is the most powerful mechanism for developing proficient software systems. It is helpful not only in declining the cost but also in the development of high quality software systems. Software developers require accurate metrics for developing efficient software system. Object-oriented metrics plays a significant role pertaining to this aspect because of their importance in the development of successful software applications. Also, the assessment of the current state of the art in metrics and object-oriented software system quality is done. Further it contains short descriptive taxonomy of the object-oriented Design and metrics. Semantic metrics [7] describes about the semantically-based metric for object-oriented systems, called the Semantic Class Definition Entropy (SCDE) metric, which examines the implementation domain content of a class to measure class complexity. The domain content is determined using a knowledge-based program understanding system. The metric's examination of the domain content of a class provides a more direct mapping between the metric and common human complexity analysis than is possible with traditional complexity measures based on syntactic aspects (software aspects related to the format of the code). Additionally, this metric represents a true design metric that can measure complexity early in the life cycles of software maintenance and software development. The SCDE metric is correlated with analyses from a human expert team, and is also compared to syntactic complexity measures. Information entropy-based metrics [1] gives the idea about Coupling of a subsystem characterizes its interdependence with other subsystems. A subsystem's cohesion, on the other hand, characterizes its internal interdependencies. When used in conjunction with other attributes, measurements of a subsystem's coupling and cohesion can contribute to software quality models. An abstraction of a software system can be represented by a graph and a module (subsystem) by a sub graph. Software design graphs depict components and their relationships. The other metrics are based on data mining [10], metrics for knowledge-base [9], aspect-oriented [11] and distributed systems [4]. Semantic metrics based on latent semantic indexing (LSI). The LSI [5] has describes a new method for automatic indexing and retrieval. The approach is to take advantage of implicit higher-order structure in the association of terms with documents (“semantic structure”) in order to improve the detection of relevant documents on the basis of terms found in queries. The particular technique used is a singular-value decomposition, in which a large term by document matrix is decomposed into a set of 100 orthogonal factors from which the original matrix can be approximated by linear combination. Documents are represented by 100 item vectors of factor weights. Queries are represented as pseudo-document vectors formed from weighted combinations of terms, and documents with supra-threshold cosine values are returned [6]. Corpus-based statistical methods are deployed for inducing and representing aspects of the meanings of words and passages reflective of their usage in large bodies of text. LSI is based on a vector space model (SVM) Vector space model (or term vector model) is an algebraic model for representing text documents (and any objects, in general) as vectors of identifiers, such as, for example, index terms. It is used in information filtering, information retrieval, indexing and relevancy rankings.

The basic objective of our work is to develop a new approach by constructing a concept relation metric to analysis the cohesion within the method. The rest of matters in this paper are as follows: Section2 describes the problem description; Section3 consists of class-method content level analysis. Section4 describes about semantic value computation. The concept relation matrix is described in Section5. Conclusion of the work has been described in Section6.

2. PROBLEM DESCRIPTION

Software cohesion can be defined as a measure of the degree to which elements of a module belong together [3]. The conceptual cohesion has major significance in software maintenance aspects [2]. In our work, we have made study of source code of a program CAccessRepots, which have been used by [12] and are available as open source. It is a small-medium size program with 51 classes and 725 distinct methods in total.

Here, we have made consideration about the class and their respective methods analysis. Also, the class-class and class- method and method–method level correlation and conceptual correlation analysis have been done in our work.
In this work, we have filtered the common method within the classes based on hierarchy level as shown in Figure 1. In each level, we have set of common methods e.g. at *level1* we have taken 51 classes and each class are having their methods. In this level we compare class1 (*c1*) method to class2 (*c2*) method and filter out the common method between them. Similarly, we fetch the common methods for class2 (*c2*) and class (*c3*), class3 (*c3*) and class4 (*c4*)... class50 (*c50*) and class51 (*c51*). The result of this level will again compare in same fashion till we found the last level common-method (*cm*) which is common to all classes. Here, we have taken the values of *leveln* common method for our consideration as they belongs to all classes and have most important as number of times occurrence in source code. The number of common-method in *levln* is twenty one and we have assumed those common-methods as *cm1, cm2, cm3… cm21* just for simplicity. The relationship between method and cm_i are shown in Table1**.**

Now, in each common-method, we have compute the specific keyword (K_i) based on their occurrence in each method. In our work, we have considered nineteen specific keywords and count of their occurrence in each common-method as shown in Table2.

Software cohesion can be defined as a measure of the degree to which elements of a module belong together [8].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.

3. CLASS-METHOD CONTENT LEVEL ANALYSIS

In this section, we have discussed about the class and their respective methods. In this context, we have 51 classes and 725 distinct methods in the considered project. We have done the common methods level analysis to filter out the common methods within the classes as shown in Figure 1. Based on the different level analysis we have found that leveln is having common methods belonging to all classes. The number of those methods is 21. We have made few assumptions for the methods.

Assumption (1): each method contains some percentage of the methods from previous class. The keywords in each method contain some percentage of keywords from all previous methods; this relation can be represented follows:

The semantic value of the keyword in the *j-th* is calculated using the occurrence of the keyword K_i and Euclidean norm as following:

For example: the relation between cm_2' and cm_2 as following:

$$cm_2' = \alpha cm_2 + (1- \alpha)\, cm_1 \quad (1)$$

Here, cm_2' is the total number of keyword in method2.
cm_2 : The number of specific keyword in method 2.
cm_1 : The total number of keyword for method 1.

And for method3

$$cm_3' = \alpha cm_3 + (1- \alpha)\, cm_2' \quad (2)$$

From equation (2)

$$cm_3' = \alpha cm_3 + (1- \alpha)\, cm_2 + (1-\alpha)^2 cm_1 \quad (3)$$

In general:

$$cm_m' = \left(\sum_{i=0}^{m-2} \alpha(1-\alpha)\right) + (1-\alpha)^{m-1} cm_{m-(m-1)} \quad (4)$$

For m=2, 3, 4…, n where, n is number of methods

Assumption2: Each keyword has a semantic value to the particular method and it may be different from method to another method. The keyword "*CString*" has the semantic values 0.2581, 0.485, in *cm2, cm6* etc. respectively. The hierarchy of semantic value for the keyword represents the importance of the keyword in corresponding common - method.

4. SEMANTIC VALUE COMPUTAION

Vector Space Model VSM has been widely used in information retrieval, information filtering, information indexing and relevancy ranking (Salton, Wong, & Yang, 1975; Tai, Ren, &Kita, 2002). In this section, we used the VSM to compute Keywords' Semantic value by constructing the Keyword-Method Matrix. In order to compute the semantic values of the keywords in each common method of source code file, every method passes through pre-processing steps to find the keywords-method list. The next step to selects the important keywords among all keywords in keywords-method list to avoid the occurrence of unrelated keywords in the final keyword-method list. The keywords (n) and their occurrence frequency are represented in the matrix A as follows:

A= K x Q keyword-method matrix consist of *K* rows (keyword) and *Q* columns (method).

This matrix represents one-row matrices and one-column matrices and these represent the vector. The Frobenius Norm of a matrix, also known as the Euclidean Norm, is defined as the square root of the sum of the absolute squares of its element, which is equal to the length of the vector.

$$f_{cm}^{j} = \sqrt{\sum_{i}^{n} K_i^2} \quad (5)$$

Where, f_{cm}^{j} is the Euclidean norm for the j-*th method cm*.

$$Kw_i = {k_i}/{f_{cm}^{i}} \quad (6)$$

Here, keyword – method matrix is computed in Table2 using the formula (5) and (6). This table consists of 19 keywords and their semantic values for twenty one methods.
The keyword common to all method with their semantic values are shown in Table2. The range of semantic value is {0-1}, which represent the importance of the keywords in the method e.g. the keyword "*InvokeHelper*" has semantic value range minimum 0.2357 for *cm1* and maximum of 0.3535 for *cm21*. Similarly, the keyword "*short*" has semantic value '0' for all method *cm1, cm2, cm3, cm4, cm6, cm7,cm9, cm10, cm11,cm12, cm13, cm14, cm16, cm18, cm20* and *cm21*, it means that this keyword in not important in those methods. This keyword "*short*" is also having semantic values 0.485, 0.485, 0.25, 0.2773 and 0.25 for *cm5, cm8, cm15, cm17* and *cm19* respectively. it means the keyword "*short*" has most important in method *cm5* and *cm8* while it has low importance in *cm15* and *cm19*.

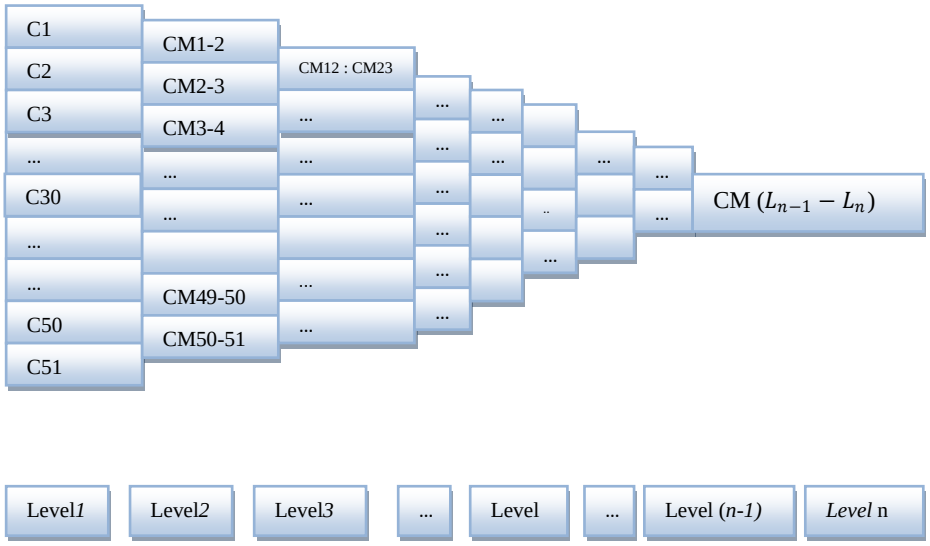

Fig 1: Hierarchy level of class and common-method

Table 1: $CM_i$and common-method relationship

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
cm1	cm2	cm3	cm4	cm5	cm6	cm7	cm8	cm9	cm10	cm11	cm12	cm13	cm14	cm15	cm16	cm17	cm18	cm19	cm20	cm21
GetApplication	GetEventProcPrefix	GetInSelection	GetIsVisible	GetLeft	GetName	GetParent	GetSection	GetTag	GetVisible	SetEventProcPrefix	SetFocus	SetInSelection	SetIsVisible	SetLeft	SetName	SetSection	SetTag	SetTop	SetVisible	SizeToFit

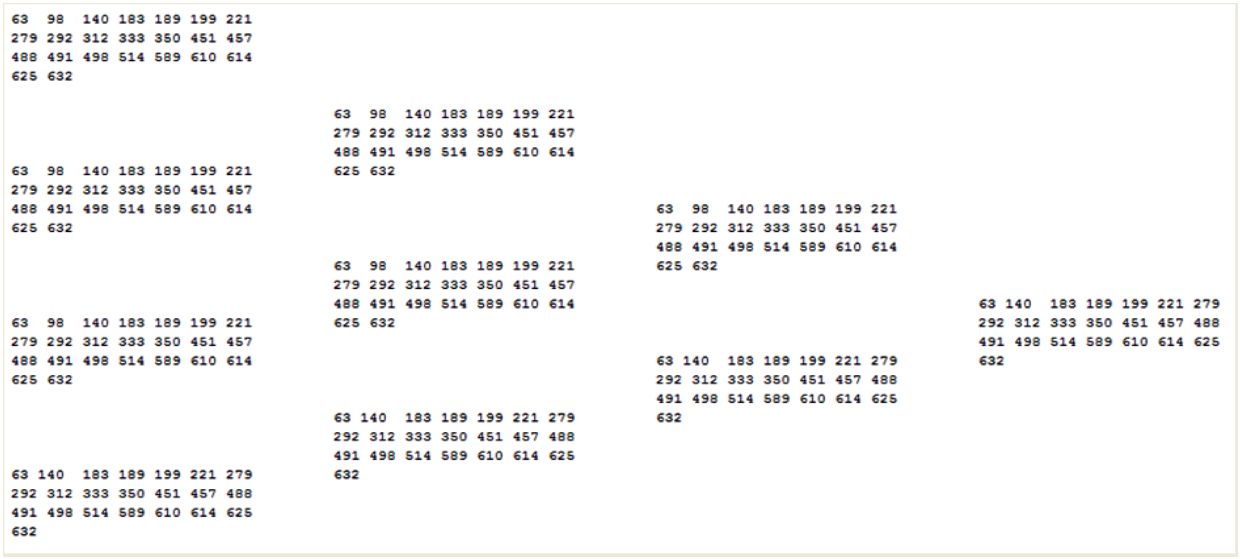

Fig 2: Class-method level hierarchy data

The class-method level hierarchy of the actual data has been shown in Figure2. We can simply correlate the Figure1 and Figure2 logically. Here, numeric values shown in Figure2 represent the method_id of methods within the classes. Since, there are 51 classes and 725 distinct method, we have shown only last four levels. On comparing the both figure1-2, we have the values of last level i.e. CM ($L_{n-1} - L_n$) are shown as in Table1 for our consideration.

5. CONCEPT RELATION MATRIX

In this section, we have used another correlation matrix such as Concept relation matrix using C.-M. Chen, [13] model has been used to justify our keyword and common-method analysis in the source code.

In Chen (2008) method, we have observed the following facts:

- Vector space model is used to estimate the concept relation degree between two course-wares in multidimensional Euclidean space.
- The importance/weight of the k-th term in the i-th courseware is calculated using TD-IDF algorithm.
- The relation r between i-th and j-th courseware with total m terms in the course units are used to build the concept relation matrix based on the weight of all keywords in i-th and j-th courseware.
- The importance/weight of the term is calculated as following:

$$w_{ik} = tf_{ik} \times \log \frac{N}{df_k} \quad (7)$$

Where, w_{ik} represents the importance/weight of the k-th term in i-th courseware.

tf_{ik} represents the term frequency of the k-th term in the i-th course ware.

df_k is the document frequency in the k-th term.

The concept relation matrix for the i-th and j-th method can be calculated using cosine-measure as following:

$$r_{ij} = \frac{\sum_{h=1}^{m} w_{ih} w_{jh}}{\sqrt{\sum_{h=1}^{m} w_{ih}^2 \sum_{h=1}^{m} w_{jh}^2}} \quad (8)$$

Where, m is the total number union terms of the i-th and j-th method.

In our work, the common-method at leveln i.e. CM at level ($L_{n-1} - L_n$) will be 21 and the relationship between each common-method with their CM_i will be shown in Table1.

Table 2: Keyword-method semantic value

										Semantic	Value										
Keywords	**cm1**	**cm2**	**cm3**	**cm4**	**cm5**	**cm6**	**cm7**	**cm8**	**cm9**	**cm10**	**cm11**	**cm12**	**cm13**	**cm14**	**cm15**	**cm16**	**cm17**	**cm18**	**cm19**	**cm20**	**cm21**
LPDISPATCH	0.4714	0	0	0	0	0	0.485	0	0	0	0	0	0	0	0	0	0	0	0	0	0
InvokeHelper	0.2357	0.2581	0.2672	0.2672	0.2425	0.2425	0.2425	0.2425	0.2425	0.2425	0.25	0.3779	0.2886	0.2886	0.25	0.25	0.2773	0.25	0.25	0.2886	0.3535
DISPATCH_PROPERTYGET	0.2357	0.2581	0	0.2672	0.2425	0.2425	0.2425	0.2425	0.2425	0.2425	0	0	0	0	0	0	0	0	0	0	0
DISPATCH_PROPERTYPUT	0	0	0.2672	0	0	0	0	0	0	0	0.25	0	0.2886	0.2886	0.25	0.25	0.2773	0.25	0.25	0.2886	0
result	1	0.7745	0.8017	0.8017	0.7276	0.7276	0.7276	0.7276	0.7276	0.7276	0	0	0	0	0	0	0	0	0	0	0
void*	0.2357	0.2581	0.2672	0.2672	0.2425	0.2425	0.2425	0.2425	0.2425	0.2425	0	0	0	0	0	0	0	0	0	0	0
CString	0	0.2581	0	0	0	0.485	0	0	0.485	0	0	0	0	0	0	0	0	0	0	0	0
BOOL	0	0	0.2672	0.2672	0	0	0	0	0	0.485	0	0	0.2886	0.2886	0	0	0	0	0	0.2886	0
short	0	0	0	0	0.485	0	0	0.485	0	0	0	0	0	0	0.25	0	0.2773	0	0.25	0	0
void	0.2357	0.2581	0	0	0	0	0	0	0	0	0.25	0.3779	0.2886	0.2886	0.25	0.25	0.2773	0.25	0.25	0.2886	0.3535
static	0	0	0	0	0	0	0	0	0	0	0.25	0	0.2886	0.2886	0.25	0.25	0.2773	0.25	0.25	0.2886	0
BYTE	0	0	0	0	0	0	0	0	0	0	0.25	0	0.2886	0.2886	0.25	0.25	0.2773	0.25	0.25	0.2886	0
lpszNewValue	0	0	0	0	0	0	0	0	0	0	0.5	0	0	0	0	0.5	0	0.5	0	0	0
DISPATCH_METHOD	0	0	0	0	0	0	0	0	0	0	0	0.3779	0	0	0	0	0	0	0	0	0.3535
VT_EMPTY	0	0	0	0	0	0	0	0	0	0	0.25	0	0.2886	0.2886	0.25	0.25	0.2773	0.25	0.25	0.2886	0.3535
nNewValue	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.5	0	0.2773	0	0.5	0	0
NULL	0	0.2581	0.2672	0.2672	0.2425	0.2425	0.2425	0.2425	0.2425	0.2425	0.25	0.7559	0.2886	0.2886	0.25	0.25	0.2773	0.25	0.25	0.2886	0.7071
LPCTSR	0	0	0	0	0	0	0	0	0	0	0.25	0	0	0	0	0.25	0	0.25	0	0	0
parms	0	0	0	0	0	0	0	0	0	0	0.5	0	0.5773	0.5773	0.5	0.5	0.5547	0.5	0.5	0.5773	0

Table 3: Keyword-method count

Keywords	cm1	cm2	cm3	cm4	cm5	cm6	cm7	cm8	cm9	cm10	cm11	cm12	cm13	cm14	cm15	cm16	cm17	cm18	cm19	cm20	cm21
LPDISPATCH	70	0	0	0	0	0	80	0	0	0	0	0	0	0	0	0	0	0	0	0	0
InvokeHelper	35	21	21	20	21	27	40	22	23	24	21	17	21	20	21	25	20	23	21	24	21
DISPATCH_PROPERTYGET	35	21	0	20	21	27	40	22	23	24	0	0	0	0	0	0	0	0	0	0	0
DISPATCH_PROPERTYPUT	0	0	21	0	0	0	0	0	0	0	21	0	21	20	21	25	20	23	21	24	0
result	105	63	63	60	63	81	120	66	69	72	0	0	0	0	0	0	0	0	0	0	0
void*	35	21	21	20	21	27	40	22	23	24	0	0	0	0	0	0	0	0	0	0	0
CString	0	21	0	0	0	54	0	0	46	0	0	0	0	0	0	0	0	0	0	0	0
BOOL	0	0	21	20	0	0	0	0	0	48	0	0	21	20	0	0	0	0	0	24	0
short	0	0	0	0	42	0	0	44	0	0	0	0	0	0	21	0	20	0	21	0	0
void	35	21	0	0	0	0	0	0	0	0	21	17	21	20	21	25	20	23	21	24	21
static	0	0	0	0	0	0	0	0	0	0	21	0	21	20	21	25	20	23	21	24	0
BYTE	0	0	0	0	0	0	0	0	0	0	21	0	21	20	21	25	20	23	21	24	0
lpszNewValue	0	0	0	0	0	0	0	0	0	0	42	0	0	0	0	50	0	46	0	0	0
DISPATCH_METHOD	0	0	0	0	0	0	0	0	0	0	0	17	0	0	0	0	0	0	0	0	21
VT_EMPTY	0	0	0	0	0	0	0	0	0	0	21	0	21	20	21	25	20	23	21	24	21
nNewValue	0	0	0	0	0	0	0	0	0	0	0	0	0	0	42	0	20	0	42	0	0
NULL	35	21	21	20	21	27	40	22	23	24	21	34	21	20	21	25	20	23	21	24	42
LPCTSR	0	0	0	0	0	0	0	0	0	0	21	0	0	0	0	25	0	23	0	0	0
parms	0	0	0	0	0	0	0	0	0	0	42	0	42	40	42	50	40	46	42	48	0

Table 4: Concept relation matrix for keyword and common method

	cm1	cm2	cm3	cm4	cm5	cm6	cm7	cm8	cm9	cm10	cm11	cm12	cm13	cm14	cm15	cm16	cm17	cm18	cm19	cm20	cm21
cm1	1	0.944	0.885	0.92	0.888	0.896	0.795	0.888	0.896	0.886	0.21	0.375	0.224	0.224	0.212	0.21	0.221	0.21	0.212	0.224	0.359
cm2	0.944	1	0.919	0.955	0.922	0.969	0.921	0.922	0.969	0.92	0.218	0.389	0.232	0.232	0.22	0.218	0.229	0.218	0.22	0.232	0.372
cm3	0.885	0.919	1	0.963	0.904	0.912	0.904	0.904	0.912	0.953	0.211	0.324	0.259	0.259	0.212	0.211	0.221	0.211	0.212	0.259	0.31
cm4	0.92	0.955	0.963	1	0.94	0.948	0.939	0.94	0.948	0.988	0.166	0.324	0.211	0.211	0.167	0.166	0.174	0.166	0.167	0.211	0.31
cm5	0.888	0.922	0.904	0.94	1	0.915	0.906	1	0.915	0.905	0.16	0.313	0.17	0.17	0.222	0.16	0.231	0.16	0.222	0.17	0.299
cm6	0.896	0.969	0.912	0.948	0.915	1	0.915	0.915	1	0.913	0.162	0.316	0.172	0.172	0.163	0.162	0.17	0.162	0.163	0.172	0.302
cm7	0.795	0.921	0.904	0.939	0.906	0.915	1	0.906	0.915	0.905	0.16	0.313	0.17	0.17	0.161	0.16	0.168	0.16	0.161	0.17	0.299
cm8	0.888	0.922	0.904	0.94	1	0.915	0.906	1	0.915	0.905	0.16	0.313	0.17	0.17	0.222	0.16	0.231	0.16	0.222	0.17	0.299
cm9	0.896	0.969	0.912	0.948	0.915	1	0.915	0.915	1	0.913	0.162	0.316	0.172	0.172	0.163	0.162	0.17	0.162	0.163	0.172	0.302
cm10	0.886	0.92	0.953	0.988	0.905	0.913	0.905	0.905	0.913	1	0.16	0.312	0.237	0.237	0.161	0.16	0.168	0.16	0.161	0.237	0.299
cm11	0.21	0.218	0.211	0.166	0.16	0.162	0.16	0.16	0.162	0.16	1	0.495	0.896	0.896	0.847	1	0.883	1	0.847	0.896	0.543
cm12	0.375	0.389	0.324	0.324	0.313	0.316	0.313	0.313	0.316	0.312	0.495	1	0.528	0.528	0.499	0.495	0.52	0.495	0.499	0.528	0.956
cm13	0.224	0.232	0.259	0.211	0.17	0.172	0.17	0.17	0.172	0.237	0.896	0.528	1	1	0.903	0.896	0.941	0.896	0.903	1	0.578
cm14	0.224	0.232	0.259	0.211	0.17	0.172	0.17	0.17	0.172	0.237	0.896	0.528	1	1	0.903	0.896	0.941	0.896	0.903	1	0.578
cm15	0.212	0.22	0.212	0.167	0.222	0.163	0.161	0.222	0.163	0.161	0.847	0.499	0.903	0.903	1	0.847	0.987	0.847	1	0.903	0.547
cm16	0.21	0.218	0.211	0.166	0.16	0.162	0.16	0.16	0.162	0.16	1	0.495	0.896	0.896	0.847	1	0.883	1	0.847	0.896	0.543
cm17	0.221	0.229	0.221	0.174	0.231	0.17	0.168	0.231	0.17	0.168	0.883	0.52	0.941	0.941	0.987	0.883	1	0.883	0.987	0.941	0.57
cm18	0.21	0.218	0.211	0.166	0.16	0.162	0.16	0.16	0.162	0.16	1	0.495	0.896	0.896	0.847	1	0.883	1	0.847	0.896	0.543
cm19	0.212	0.22	0.212	0.167	0.222	0.163	0.161	0.222	0.163	0.161	0.847	0.499	0.903	0.903	1	0.847	0.987	0.847	1	0.903	0.547
cm20	0.224	0.232	0.259	0.211	0.17	0.172	0.17	0.17	0.172	0.237	0.896	0.528	1	1	0.903	0.896	0.941	0.896	0.903	1	0.578
cm21	0.359	0.372	0.31	0.31	0.299	0.302	0.299	0.299	0.302	0.299	0.543	0.956	0.578	0.578	0.547	0.543	0.57	0.543	0.547	0.578	1

The values of Table2 have been calculated by using formula (5), (6) and data of Table3 e.g. the value of cm_1 of "*LPDISPATCH*" keyword can be calculated as follows:

Step1:

$$f_{cm}^{1} = \sqrt{\sum_{i=1}^{19}(K_1^2 + K_2^2 + \cdots + K_{19}^2)}$$

$$= 148.492424049175$$

Step2:

$$Kw_1 = {}^{k_1}\Big/_{f_{cm}^{1}} = 70/148.492424049175$$

$$= 0.4714$$

In Table4, we have computed the concept relation matrix for our keyword-method using Chen (2008) model. We have computed those values by using formula (7), (8) and data of Table3 e.g. the value of CM_1-CM_1 can be calculated as follows:

Step1:

$$w_{ik} = tf_{ik} \times \log \frac{N}{df_k}$$

$$= 70*\log\frac{21}{150} = -59.771$$

Step 2:

$$r_{ij} = \frac{\sum_{h=1}^{m} w_{ih} w_{jh}}{\sqrt{\sum_{h=1}^{m} w_{ih}^2 \sum_{h=1}^{m} w_{jh}^2}}$$

$$r_{ij} = \frac{\sum_{h=1}^{19} w_{i19} w_{j19}}{\sqrt{\sum_{h=1}^{19} w_{i19}^2 \sum_{h=1}^{19} w_{j19}^2}}$$

For i = 1 and j = 1,

$\sum_{h=1}^{19} w_{i19} w_{j19}$ =39474.03

$\sqrt{\sum_{h=1}^{19} w_{i19}^2 \sum_{h=1}^{19} w_{j19}^2} = \sqrt{1558198669}$

39474.03

$r_{11} = \frac{39474.03}{39474.03}$ =1

Similarly, we can compute for all remaining row-column values for Table4.

6. CONCLUSION

A heuristic method for class and method relationship sequencing for software maintenance has been developed and implemented using source code program and filtering techniques. The semantic value of the keyword is obtained based on the importance of the keywords in the method at the different levels of extraction. These semantic values are used to find the relation between the methods by computing different semantic values for all methods. Another model used the frequency appearance of the keywords, and these methods ignored the importance of the keyword in different method which we have kept in our computation of the proposed heuristic method. Other researchers have assigned the weights directly to the class-method whereas in our work, we have assigned semantic values to the keywords in the method. As a scope of future work, apart from taking the common keywords between class-methods, the semantic value of the important keywords in each class-method can be considered to find the cohesion among the classes. The computation of correlation matrix can be calculated with other methods such as Hsieh and Wang (2010) and Sami and Mishra (2011) to select the best computation method for the purpose. This method is further utilized in perfective maintenance task in determination of coupling of cohesion.

7. REFERENCES

[1] Allen, E.B., Khoshgoftaar T.M., and Chen Y., 2001. Measuring Coupling and Cohesion of Software Modules: An Information-Theory Approach,". In Proceedings of Seventh IEEE Int'l Software Metrics Symp.

[2] Andrian Marcus, *member,IEEE computer society*, Denys P. 2007 Using the conceptual cohesion of classes for fault prediction in object-oriented system.

[3] Bieman, J., and Kang, B.-K. ,1995. Cohesion and Reuse in an Object-Oriented System. Proceedings of Symp. Software Reusability.

[4] Counsell,S., Swift,S. and Tucker,A., 2005. Object-Oriented Cohesion as a Surrogate of Software Comprehension: An Empirical Study, Proc. Fifth IEEE Int'l Workshop Source Code Analysis and Manipulation.

[5] Deerwester, S., Dumais, S.T., Furnas, G.W.,Landauer, T.K. and Harshman, R., 1990. Indexing by Latent Semantic Analysis, J. Am. Soc. Information Science, vol. 41, pp. 391-407.

[6] Dumais,S.T., 1991. Improving the Retrieval of Information from External Sources, Behavior Research Methods, Instruments, and Computers, vol. 23, no. 2, pp. 229-236.

[7] Etzkorn,L.H.,Gholston, S., and Hughes,W.E., 2002. A Semantic Entropy Metric, J. Software Maintenance: Research and Practice, vol. 14, no. 5, pp. 293-310.

[8] Briand,L.C.,Daly, J.W., and st,J. Wu¨, 1998. A Unified Framework for Cohesion Measurement in Object-Oriented Systems, Empirical Software Eng., vol. 3, no. 1, pp. 65-117.

[9] Kramer,S., and Kaindl,H., 2004. Coupling and Cohesion Metrics for Knowledge-Based Systems Using Frames and Rules, ACM Trans. Software Eng. and Methodology, vol. 13, no. 3, pp. 332-358.

[10] Montes de Oca, C., and Carver, D.L., 1998. Identification of Data Cohesive Subsystems Using Data Mining Techniques, Proc. 14th IEEE Int'l Conf. Software Maintenance, pp. 16-23.

[11] Zhao,J., and Xu,B., 2004. Measuring Aspect Cohesion, Proc. Seventh Int'l Conf. Fundamental Approaches to Software Eng., pp. 54-68.

[12] Kanellopoulos, Y. and Tjortjis, C., 2004. Data Mining Source Code to Facilitate Comprehension: Experiments on Clustering Data Retrieved from C++ Program, Proceedings of the 12th IEEE International Workshop on Program Comprehension (IWPC'04).

[13] Chen, C.-M., 2008. Intelligent web-based learning system with personalized learning path guidance. Computers & Education, 51(2), 787–814.

[14] Chen, P.-I., Lin, S.-J., & Chu, Y.-C., 2011. Using Google latent semantic distance to extract the most relevant information. Expert Systems with Applications, 38(6), 7349–7358.

[15] Shirabad,J. S., Lethbridge T. C. and Matwin, S., 2003. Mining the Maintenance History of Legacy Software System, Proceedings of the International Conference on Software Maintenance (ICSM'03), IEEE.

[16] Hsieh, T.-C., and Wang, T.-I. (2010). A mining-based approach on discovering courses pattern for constructing suitable learning path. Expert Systems with Applications, 37(6), 4156–4167.

[17] Al-Radeai, M.,Sami and Mishra R.,B (2011). A heuristic method for learning path sequencing for intelligent tutoring system in E-learning. Int. J. of Intelligent Information System, vol. (7), Issue (4).