Volume 3, Issue 3, March 2013 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

A Novel Approach for Achieving High Cohesion and Fault Calculation in Object Oriented Systems by Using C3 Metrics

S.Shanmugapriya*,M.Monisha Devi
Assistant Professor
Department Of Computer science & Engineering
Veltech Hightech Dr.Rangarajan Dr.Sakunthala Engineering College

Abstract— Software modularization, Object-Oriented (OO) decomposition in particular, is an approach for improving the organization and comprehension of source code. Software cohesion can be defined as a measure of the degree to which elements of a module belong together. Currently proposed measures for cohesion in Object-Oriented (OO) software reflect particular interpretations of cohesion and capture different aspects of it. 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 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. The measure, named the Conceptual Cohesion of Classes (C3), is inspired by the mechanisms used to measure textual coherence in cognitive psychology and computational linguistics. This paper presents the principles and the technology that stand behind the C3 measure.

Keywords— Software cohesion, textual coherence, fault prediction, fault proneness, Latent Semantic Indexing.

I. INTRODUCTION

In order to understand OO software, software engineers need to create a well-connected representation of the classes that make up the system. Each class must be understood individually and, then, relationships among classes as well. 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. Such definitions, although very intuitive, are quite vague and make cohesion measurement a difficult task, leaving too much room for interpretation. In OO software systems, cohesion is usually measured at the class level and many different OO cohesion metrics have been proposed which try capturing different aspects of cohesion or reflect a particular interpretation of cohesion 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. Cohesion is usually measured on structural information extracted solely from the source code (for example, attribute references in methods and method calls) that captures the degree to which the elements of a class belong together from a structural point of view. Thus, they give no clues as to whether the class is cohesive from a conceptual point of view nor do they give an indication about the readability, other types of metrics were proposed by researchers to capture different aspects of cohesion, only a few such metrics address the conceptual and textual aspects of cohesion . This paper 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. The conceptual relation between methods is based on the principle of textual coherence. The source code is far from a natural language and many aspects of natural language discourse do not exist in the source code or need to be redefined. The rules of discourse are also different from the natural language. C3 is based on the analysis of textual information in the source code, expressed in comments and identifiers. Once again, this part of the source code, although closer to natural language, is still different from it. Thus, using classic natural language processing methods. Hence, we use an Information Retrieval (IR) technique, namely, Latent Semantic Indexing (LSI), to extract, represent, and analyse the textual information from the source code. Our measure of cohesion can be interpreted as a measure of the textual coherence of a class within the context of the entire system. Cohesion ultimately affects the comprehensibility of source code.

II. RELATED WORK: COHESION MEASURES FOR OO SOFTWARE SYSTEMS

There are several different approaches to measure cohesion in OO systems. Many of the existing metrics are adapted from similar cohesion measures for non-OO systems (we are not discussing those here), while some of the metrics are specific to OO software. Based on the underlying information used to measure the cohesion of a class, one can distinguish structural metrics [8], [11], [20], semantic metrics information entropy-based metrics [1], slice-based metrics, metrics based on data mining, and metrics for specific types of applications like knowledge-based, aspect-oriented, and distributed systems. The class of structural metrics is the most investigated category of cohesion metrics and includes lack of cohesion in methods.

LCOM1, LCOM3, LCOM4, C_o (connectivity), LCOM5, TCC (tight class cohesion) [8], LCC (loose class cohesion) [8], ICH (information-flow-based cohesion), NHD (normalized Hamming distance), etc. Cohesion is seen to be dependent on the number of pairs of methods that share instance or class variables one way or another. The differences among the structural metrics are based on the definition of the relationships among methods, system representation, and counting mechanism. A comprehensive overview of graph theory-based cohesion metrics. Somewhat different in this class of metrics are LCOM5 and Cohesion, which consider that cohesion is directly proportional to the number of instance variables in a class that are referenced by the methods in that class defined a unified framework for cohesion measurement in OO systems [11] which classifies and discusses all of these metrics. A small set of cohesion metrics was proposed for specific types of applications. From a measuring methodology point of view, two other cohesion metrics are of interest here since they are also based on an IR approach. However, IR methods are used differently there than in our approach proposed a composite cohesion metric that measures the information strength of a module. This measure is based on a vector representation of the frequencies of occurrences of data types in a module. The approach measures the cohesion of individual subprograms of a system based on the relationships to each other in this vector space. A file-level cohesion metric based on the same type of information that we are using for our proposed metrics here. The designers and the programmers of a software system often think about a class as a set of responsibilities that approximate the concept from the problem domain implemented by the class as opposed to a set of method-attribute interactions. Information that gives clues about domain concepts is encoded in the source code as comments and identifiers.

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 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. In order to extract and analyse the unstructured information from the source code, we use LSI, which is an advanced IR method. Although the general approach would work with other IR methods or with more complex natural language processing techniques, we decided to use LSI here for several reasons. First, we have extensive experience in using LSI for other software engineering problems like concept and feature location, traceability link recovery between source code and documentation, identification of abstract data types in legacy source code and clone detection. In addition to other usages in software engineering, LSI has been successfully used in cognitive psychology for the measurement of textual coherence, which is the principle upon which we base our approach.

A. OVERVIEW OF LATENT SEMANTIC INDEXING

LSI is a corpus-based statistical method for inducing and representing aspects of the meanings of words and passages (of the natural language) reflective of their usage in large bodies of text. LSI is based on a vector space model (VSM) as it generates a real-valued vector description for documents of text. Results have shown [7], that LSI captures significant portions of the meaning not only of individual words but also of whole passages, such as sentences, paragraphs, and short essays. LSI was originally developed in the context of IR as a way of overcoming problems with and synonymy that occurred with VSM approaches. Some words appear in the same contexts and an important part of word usage patterns is blurred by accidental and inessential information. The method used by LSI to capture the essential semantic information is dimension reduction, selecting the most important dimensions from a co-occurrence matrix (words by context) decomposed using singular value decomposition (SVD).

B. MEASURING TEXT COHERENCE WITH LATENT SEMANTIC INDEXING

In a language such as English, there are many aspects of a discourse that contribute to coherence, including co reference, causal relationships, connectives, and signals. Existing approaches in cognitive psychology and computational linguistics for automatically measuring text coherence are based on propositional modelling. LSI can be applied as an automated method that produces coherence predictions similar to propositional modelling. The primary method for using LSI to make coherence predictions is to compare some unit of text to an adjoining unit of text in order to determine the degree to which the two are semantically related. These units could be sentences, paragraphs, individual words, or even whole books. Coherence predictions have typically been performed at a propositional level in which a set of propositions all contained within the working memory are compared or connected to each other. For an LSI-based coherence analysis, using sentences as the basic unit of text appears to be an appropriate corresponding level that can be easily parsed by automated methods. Sentences serve as a good level in that they represent a small set of textual information (for example, typically three to seven propositions) and thus would be approximately consistent with the amount of information that is held in short-term memory. To measure the coherence of a text, LSI is used to compute the similarities between consecutive sentences in the text. High similarity between two consecutive sentences indicates that the two sentences are related, while low similarity indicates a break in the topic. The overall coherence of a text is measured as the average of all similarity measures between consecutive sentences.

C. FROM TEXTUAL COHERENCE TO SOFTWARE COHESION

We adapt the LSI-based coherence measurement mechanism to measure cohesion in OO software. One issue is the definition of documents in the corpus. For a natural language, sentences, paragraphs, and even sections are used as units of text to be indexed (that is, documents). Based on our previous experience, we consider methods as elements of the source code that can be units for indexing. Thus, the implementation of each method is converted to a document in the corpus to be indexed by LSI. Another issue of interest lies in the extraction of relevant information from the source code. We extract all identifiers and comments from the source code. In software, we interpret a cohesive class as implementing one concept (or a very small group of related concepts) from the software domain. With that in mind, each method of the class will refer to some aspect related to the implemented concept. Hence, methods should be related to each other conceptually. 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.

1. **Corpus creation.**
   The source code is pre-processed 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.
2. **Corpus indexing**
   LSI is used to index the corpus and create an equivalent semantic space.
3. **Computing conceptual similarities.**
   Conceptual similarities are computed between each pair of methods.
4. **Computing C3.**
   Based on the conceptual similarity measures, C3 is computed for each class.

IRC^{3}M is implemented as an MS Visual Studio .NET add-in 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. We use the cosine between vectors in the LSI space to compute conceptual relations.

D. THE CONCEPTUAL COHESION OF CLASSES

In order to define and compute the C3 metric, we introduce a graph-based system representation similar to those used to compute other cohesion metrics. We consider an OO system as a set of classes C = {c1; c_2 ... *cn}*. The total number of classes in the system C is n = |C|. A class has a set of methods. For each class c € C, M(c) = {m_1.....; mk*}* is the set of methods of class c.

An OO system C is represented as a set of connected graphs G_C = {$G_{1;...;}$ G_n} with Gi representing class c_i. Each class *ci* € C is represented by a graph Gi € G_C such that Gi = (Vi; Ei), where Vi = M(ci) is a set of vertices corresponding to the methods in class q, and Ei € Vi x Vi is a set of weighted edges that connect pairs of methods from the class.

DEFINITION 1

(CONCEPTUAL SIMILARITY BETWEEN METHODS--(CSM)).

For every class c_i 2 C, all of the edges in E_i are weighted. For each edge (mk ;mj) € Ei, we define the weight of that edge CSM(mk ;mj) as the conceptual similarity between the methods mk and mj The conceptual similarity between two methods *mk* and *mj*, that is, CSM(m_k *;mj)*, is computed as the cosine between the vectors corresponding to *mk* and *mj* in the semantic space constructed by the IR method.

$$CSM(mk, mj) = vm_k^t \, vm_i / |vm_k|_2 \, |vm_i|_2$$

Where vm_k and *vmj* are the vectors corresponding to the mk; mj 2 M (Ci) methods, T denotes the transpose, and vmkj $_2$ is the length of the vector. For each class c 2 C, we have a maximum of N = C2 distinct edges between different nodes, where n = |Mc)|. With this system representation, we define a set of measures that approximate the cohesion of a class in an OO software system by measuring the degree to which the methods in a class are related conceptually.

DEFINITION 2

(AVERAGE CONCEPTUAL SIMILARITY OF METHODS IN A CLASS (ACSM)).

The ACSM c € C is

$$ACSM(c) = 1/N \times \sum CSM\,(mi\ ,mj)$$

Where (mi; mj) € E, i ≠ j, mi , mj€ M(c), and N is the number of distinct edges in G, as defined in Definition 1.

Table: 1 Conceptual Similarity between the Methods in the example Class

		m1	m2	m3	m4
m1	CanCreateWrapper	1	0.971	0.968	0.889
m2	CanCreateInstance		1	0.995	0.828
m3	CanGetService			1	0.827
m4	CanAccess				1

C3(Example) = 0:913.

DEFINITION 3 (C3).

For a class c € C, the conceptual cohesion of, C3(c) is defined as follows

C3(c) = {ACSM(c), if ACSM(c)>0,

Based on the above definitions, C3 (c) € [0, 1] V c € C. If a class c 2 C is cohesive, then C3(c) should be closer to one, meaning that all methods in the class are strongly related conceptually with each other (that is, the CSM for each pair of methods is close to one).

IV. CONCLUSION & FUTURE WORK

Classes in object-oriented systems, written in different programming languages, contain identifiers and comments which reflect concepts from the domain of the software system. Given our results that show that C3 and some structural metrics complement each other at least for fault prediction, more insights into the combination of conceptual and structural metrics for solving other problems are required and planned. Specifically, we intend to address crosscutting concerns, which affect the cohesion of classes. We are planning user studies to test whether C3 may reflect the perception of the programmers on class cohesion. Finally, we plan to test this approach on several releases of a system such that the models are built on a given release and faults are predicted for the next release of the 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. This paper defines the conceptual cohesion of classes, which captures new and complementary dimensions of cohesion compared to a host of existing structural metrics. Principal component analysis of measurement results on three open source software systems statistically supports this fact. In addition, the combination of structural and conceptual cohesion metrics defines better models for the prediction of faults in classes than combinations of structural metrics alone. Highly cohesive classes need to have a design that ensures a strong coupling among its methods and a coherent internal description.

REFERENCES:

[1] E.B. Allen, T.M. Khoshgoftaar, and Y. Chen, "Measuring Coupling and Cohesion of Software Modules: An Information-Theory Approach," Proc. Seventh IEEE Int'l Software Metrics Symp. pp. 124-134, Apr. 2001
[2] G. Antoniol, G. Canfora, G. Casazza, and A. De Lucia, "Identifying the Starting Impact Set of a Maintenance and Reengineering," Proc. Fourth European Conf. Software Maintenance, pp. 227-230, 2000.
[3] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo, "Recovering Traceability Links between Code and Documentation," IEEE Trans. Software Eng., vol. 28, no. 10, pp. 970-983, Oct. 2002.
[4] E. Arisholm, L.C. Briand, and A. Foyen, "Dynamic Coupling Measurement for Object-Oriented Software," IEEE Trans. Software Eng., vol. 30, no. 8, pp. 491-506, Aug. 2004.
[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] 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, Oct. 1996.
[7] M.W. Berry, "Large Scale Singular Value Computations," Int'l J. Supercomputer Applications, vol. 6, pp. 13-49, 1992.
[8] J. Bieman and B.-K. Kang, "Cohesion and Reuse in an Object- Oriented System," software Reusability, Apr. 1995.
[9] L. Briand, W. Melo, and J. Wust, "Assessing the Applicability of Fault-Proneness Models Across Object-Oriented Software Projects," IEEE Trans. Software Eng., vol. 28, no. 7, pp. 706-720, July 2002.
[10] L.C. Briand, J.W. Daly, V. Porter, and J. Wu¨ st, "A Comprehensive Empirical Validation of Design Measures for Object-Oriented Systems," Proc. Fifth IEEE Int'l Software Metrics Symp., pp. 43-53, Nov. 1998.
[11] L.C. Briand, J.W. Daly, and J. Wu¨ st, "A Unified Framework for Cohesion Measurement in Object-Oriented Systems," Empirical Software Eng., vol. 3, no. 1, pp. 65-117, 1998.
[12] L.C. Briand, S. Morasca, and V.R. Basili, "Property-Based Software Engineering Measurements," IEEE Trans. Software Eng., vol. 22, no. 1, pp. 68-85, Jan. 1996.
[13] 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..
[14] F. Brito e Abreu and M. Goulao, "Coupling and Cohesion as Modularization Drivers: Are We Being Over-Persuaded," Proc. Fifth European Conf. Software Maintenance and Reeng., pp. 47-57, 2001.
[15] H.S. Chae, Y.R. Kwon, and D.H. Bae, "A Cohesion Measure for Object-Oriented Classes," Software: Practice and Experience, vol. 30, pp. 1405-1431, 2000.
[16] 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.

[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] S. Chidamber, D. Darcy, and C. Kemmerer, "Managerial Use of Metrics for Object-Oriented Software: An Exploratory Analysis," IEEE Trans. Software Eng., vol. 24, no. 8, pp. 629-639, Aug. 1998.
[19] S.R. Chidamber and C.F. Kemmerer, "Towards a Metrics Suite for Object-Oriented Design," Proc. Sixth ACM Conf. Object-Oriented Programming, Systems, Languages and Applications, pp. 197-211, 1991.
[20] S.R. Chidamber and C.F. Kemmerer, "A Metrics Suite for Object- Oriented Design," IEEE Trans. Software Eng., vo l. 20, no. 6, pp. 476- 493, June 1994.

