Conceptual Cohesion of Classes(C3) Metrics

Girish K. K.[1]

M. Tech in Software Engineering, Department of Computer science and Engineering (PG)
Sarabhai Institute of Science and Technology (SIST), Trivandrum, India

Abstract: ***High 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. 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. An open source eclipse plug-in is developed as part of this research work to demonstrate how well cohesion of classes is predicted using C3 metrics.***

Keywords: Software cohesion, textual coherence, fault prediction, fault proneness, program comprehension, information retrieval, Latent Semantic Indexing.

1. Introduction

It is widely accepted that object oriented development requires a different way of thinking than traditional structured development and software projects are shifting to object oriented design. The main advantage of object oriented design is its modularity and reusability. Object oriented metrics are used to measure properties of object oriented designs. Metrics are a means for attaining more accurate estimations of project milestones, and developing a software system that contains minimal faults. Project based metrics keep track of project maintenance, budgeting etc. Design based metrics describe the complexity, size and robustness of object oriented and keep track of design performance.

Software modularization, Object-Oriented (OO) decomposition in particular, is an approach for improving the organization and comprehension of source code. 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. 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 belongs together. 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.

Proposals of measures and metrics for cohesion abound in the literature as software cohesion metrics proved to be useful in different tasks , including the assessment of design quality, productivity, design, and reuse effort, prediction of software quality, fault prediction, modularization of software, and identification of reusable of components .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. The trade- off is that such measures deal with information that can be automatically extracted from software and analyzed by automated tools and ignore less structured but rich information from the software (for example, textual information). 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. These measures give information about the way a class is built and how its instances work together to address the goals of their design. The principle behind this class of metrics is to measure the coupling between the methods of a class. Thus, they give no clues as to whether the class is cohesive from a conceptual point of view (for example, whether a class implements one or more domain concepts) nor do they give an indication about the readability and comprehensibility of the source code.

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. These measures give information about the way a class is built and how its instances work together to address the goals of their design. The principle behind this class of metrics is to measure the coupling between the methods of a class. Thus, they give no clues as to whether the class is cohesive from a conceptual point of view (for example, whether a class implements one or more domain concepts) nor do they give an indication about the readability and comprehensibility of the source code.

Although other types of metrics were proposed by researchers (SCDE, LORM, PSI etc) to capture conceptual

aspects of cohesion, only a few such metrics use good and efficient algorithms for information retrieval from unstructured (comments and identifiers) data in the source code. From the literature analysis, a need for an efficient and pertinent metrics for conceptual cohesion arises undoubtedly, supported by an efficient Information Retrieval (IR) algorithm.

This paper propose a new measure for class cohesion, named the Conceptual Cohesion of Classes (C3) [7], 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. We interpret the implementation of methods as elements of discourse. There are many aspects of a discourse that contribute to coherence, including co-reference, causal relationships, connectives, and signals. 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, such as propositional analysis, is impractical or unfeasible. Hence, we use an Information Retrieval (IR) technique, namely, Latent Semantic Indexing (LSI), to extract, represent, and analyze 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.

1.1 Problem definition

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. Among the existing cohesion metrics for OO software, the Logical Relatedness of Methods (LORM) and the Lack of Conceptual Cohesion in Methods (LCSM) are the only ones that use this type of information to measure the conceptual similarity of the methods in a class. The philosophy behind this class of metrics, into which this work falls, is that a cohesive class is a crisp implementation of a problem or solution domain concept. Hence, if the methods of a class are conceptually related to each other, the class is cohesive. From the above understanding, the hypothesis is " *how to utilize standard programming practices, especially comments and proper identifier names in measuring software cohesion of classes* ".

The remainder of this paper is organized as follows: Section 2 describes the analysis of c3 metrics; Section 3 describes the related works. Section 4 gives the conclusion.

2. Analysis of C3 Metrics

For the purpose of demonstrating the effectiveness of the C3 metrics, an open source eclipse plug-in is developed called the C3 Metrics tool. As the input to the software, user feeds the object oriented source code file. This step is very essential, as the source code file forms the basis for the calculation of C3 metric. It is from the source code file that the tool extracts keywords for the calculation of C3 metrics. The source code should follow all the universally accepted industrial standards and naming conventions as well as proper commenting. Identifier names and comments forms the domain concept behind the source code. Reasonable naming conventions and proper commenting are the driving factors in the calculation of C3 metrics. So user is strictly recommended to follow industrial coding standards in developing source code.

As the source code file is an object oriented one, the system first extract classes from the source code. As class cohesion is our objective, extraction of classes is very essential. For this, any parser can be used in parsing the source code. The parser will identify which all are the classes in source code, and extract the classes. The parser should extract not only the classes in the source code, but also the methods inside the classes. Method extraction is very important. Conceptual class cohesion is predicted as how well methods of a class are related to each other conceptually. So method extraction along with class is very important.

The system will extract comments and identifier names corresponding to each methods extracted previously to produce relevant keywords. Irrelevant keywords are discarded. A document in the corpus is created for each method in every class. This is done by generating a matrix with columns representing documents and rows representing each terms. We call this matrix as term-document matrix. The term-document matrix is simply a matrix with all the terms in a collection on one axis and all the document on other. If a term i appears one or more times in a document j, then the value of i,jth element of the term-document matrix is 1, otherwise 0. Latent Semantic Indexing (LSI) is applied on the term-document matrix.

The metrics calculation phase includes the actual calculation of C3 metrics based on the output received from the Information Retrieval phase. In order to define and compute the C3 metric, i am introducing 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, c2,.... cn } The total number of classes in the system C is n = |C|. A class has a set of methods. For each class c ϵ C, M(c) = { m1, m2,... mk } is the set of methods of class c.

An OO system C is represented as a set of connected graphs $Gc = \{G1, G2, \ldots Gn\}$, with Gi representing class ci . Each class ci ϵ C is represented by a graph Gi ϵ Gc such that $Gi = (Vi, Ei)$, where Vi = M(ci) is a set of vertices corresponding to the methods in class ci , and Ei C Vi x Vi is a set of weighted edges that connect pairs of methods from the class.

a. Conceptual Similarity between Methods (CSM)

For every class ci ϵ C, all of the edges in Ei are weighted. For each edge (mk, mj) ϵ Ei , we define the weight of that edge CSM(mk, mj) as the conceptual similarity between the methods mk and mj .

$$CSM(mk, mj) = \frac{vm_k^T vm_j}{|vm_k|2 * |vm_j|2}$$

b. Average Conceptual Similarity of Methods (ACSM)

ACSM(c) defines the degree to which methods of a class belong together conceptually and, thus, it can be used as a basis for computing the C3.
The ACSM c ϵ C is

$$ACSM(c) = \frac{1}{N} * \sum_{i=1}^{N} CSM(m_i, m_j)$$

c. Conceptual Cohesion of Classes (C3)

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

$$C3(c) = \begin{cases} ACSM(c), \text{if } ACSM(c) > 0, \\ \text{else}, 0 \end{cases}$$

2.1 Latent semantic indexing (ISI)

Latent Semantic Analysis [8] arose from the problem of how to find relevant documents from search words. The fundamental difficulty arises while comparing words to find relevant documents, because what we really want to do is compare the meanings or concepts behind the words. LSA attempts to solve this problem by mapping both words and documents into a concept space and doing the comparison in this space.LSI uses a technique called Singular Value Decomposition (SVD) by which a term- document matrix is divided into three other matrixes - a term matrix, a singular matrix and a document matrix. Initial term-document matrix contains terms on one axis and documents on other. A cell in the matrix denoted occurrence of a particular term in document. After taking the dot product of the three matrices we will get vector space for the document.

Latent Semantic Indexing (LSI) is a method for discovering hidden concepts in document data. Each document and term (word) is then expressed as a vector with elements corresponding to these concepts. Each element in a vector gives the degree of participation of the document or term in the corresponding concept. The goal is not to describe the concepts verbally, but to be able to represent the documents and terms in a unified way for exposing document-document, document-term, and term-term similarities or semantic relationship which is otherwise hidden.

LSA can be viewed as both a model of the underlying representation of knowledge and its acquisition or as a practical method for estimating aspects of similarities in meaning. As a model of knowledge, the coherence predictions are similar to those of propositional modeling. One can think of our experience of coherence as being an effect of computing semantic relationships between pieces of textual information. These semantic relationships are based on our exposure to this information in the past. Through our experiences of words co-occurring, or occurring in similar contexts, we develop knowledge structures which capture these relationships. The LSA coherence predictions model both the effects of co-reference and also the semantic relatedness as measured by the analysis of contextual occurrences in the past. In this case, the past experiences are based on a set of initial training texts.

As a practical method, LSA produces a useful representation for text research. The ability to measure text-to-text relationships permits predictions of human judgments of similarity. These judgments are based not only direct term co-occurrence but also a deeper measure of inferred semantic relationships based on past contextual experiences. The results from the analyses described in this paper indicate that LSA captures to a large degree the variable coherence of texts which correlate highly with readers' actual comprehension of the texts. Since the method is automatic, it permits rapid analyses of texts, thereby avoiding some of the effort involved in performing propositional analyses and allowing analyses that could not have been performed previously. LSA can be conceived as being both a model of the representation of knowledge and a practical method.

LSI overcomes two of the most problematic constraints of Boolean keyword queries: multiple words that have similar meanings (synonymy) and words that have more than one meaning (polysemy). Synonymy is often the cause of mismatches in the vocabulary used by the authors of documents and the users of information retrieval systems. As a result, Boolean or keyword queries often return irrelevant results and miss information that is relevant. LSI is also used to perform automated document categorization. In fact, several experiments have demonstrated that there are a number of correlations between the way LSI and human's process categorize text.

3. Related Works

Chidamber and Kemerer (CK) metric [1],[2] suite is a structural metric suite for object oriented system. These metrics are based on measurement theory and also reflects the viewpoints of experienced OO software developers. Chidamber and Kemerer (CK) metric suite is completely related to the structural aspects of the source code. The Chidamber & Kemerer metrics suite originally consists of 6 metrics calculated for each class. They are, Weighted Methods per Class (WMC), Depth of Inheritance Tree (DIT), Number of Children (NOC), coupling between Object classes (CBO), Response for a Class (RFC), Lack of Cohesion in Methods (LCOM). CK metric suites are particularly designed for the structural aspects of class cohesion. They do not provide a better understanding of whether a class is conceptually cohesive.

Semantic Class Definition Entropy (SCDE) [5] metrics introduces a semantically based metric for object oriented systems. SCDE metrics measures the complexity of classes in OO software, using domain related information from class documentation (comments and identifiers). In order to retrieve unstructured (comments and identifiers) information from the source code, SCDE uses an information retrieval

technique called PATRicia (Program Analysis Tool for Reuse).

LOgical Relatedness of Method (LORM) [4] measures logical cohesion of a class in OO software. Most desirable cohesion is the model cohesion in which class represents a single semantically meaningful concept. To determine whether a class represents a single semantically meaningful concept, the LORM metrics measures the conceptual relatedness of methods of the class as determined by the understanding of the class methods represented by a semantic network of conceptual graphs.

Percentage of Shared Ideas (PSI) [9] is a metric for measuring the semantic cohesion of a class in object-oriented software. PSI uses information in a knowledge base to quantify the cohesiveness of a class's task in the problem domain, allowing a clearer view of cohesion than code syntax provides. Furthermore, this metric is independent of code structure and could be calculated before implementation, providing clues to design flaws earlier in the software development cycle, when changes are less expensive.

4. Conclusion

The hypothesis is "how to utilize standard programming practices, especially comments and proper identifier names in measuring software cohesion of classes ". The experiments and findings do support the hypothesis. 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.

This paper defines the conceptual cohesion of classes, which captures new and complementary dimensions of cohesion compared to a host of existing structural metrics. My major findings are,

1) Conceptual cohesion is a highly advantageous, most relevant aspect of software cohesion which was concealed due to overwhelming usage of structural cohesion.
2) Current industrial settings need a powerful software cohesion metrics that can save software developers as well as software testers bug identification time and debugging time.
3) Most of the structural metrics used today will not incorporate an important aspect of object oriented concept- objects interrelate real world scenarios, which can be solved using Conceptual Cohesion of Classes(C3) metrics.
4) Conceptual Class Cohesion emphasis on the object (real world scenario) and the object itself is the 'concept' in Conceptual cohesion.
5) Comments and identifier names, even though appears static in a source code file, are the best source of information supply. In other words we can say that they are the whole sale suppliers of information.
6) Latent Semantic Indexing (LSI) without any substitute is the most efficient information retrieval approaches available today.

I conclude based upon the aforementioned findings and understandings that a Conceptual Cohesion Measuring tool can help software developers as well as software testers and may revolutionalize the IT sector, the software engineering field and research field in near future.

5. Acknowledgment

This research study was supported by the Department of Computer Science, Faculty of Software Engineering, Sarabhai Institute of Science and Technology. I wish to thanks to Prof. Dr. C. G. Sukumaran Nair, Head of CSE Department and Ass. Prof. Mrs. Lizmol Stephen, my project guide, for their invaluable support, encouragement, supervision and useful suggestions throughout my work.

References

[1] S.R. Chidamber and C.F. Kemerer, "A Metrics Suite for Object- Oriented Design," IEEE Trans. Software Eng., vol. 20, no. 6, pp. 476-493, June 1994.
[2] D. Darcy and C. Kemerer, "OO Metrics in Practice," IEEE Software,vol. 22, no. 6, pp. 17-19, Nov./Dec. 2005.
[3] S. Deerwester, S.T. Dumais, G.W. Furnas, T.K. Landauer, and R. Harshman, "Indexing by Latent Semantic Analysis," J.Am. Soc.Information Science, vol.41, pp.391-407, 1990.
[4] L. Etzkorn and H. Delugach, "Towards a Semantic Metrics Suite for Object-Oriented Design," Proc. 34th Int'l Conf. Technology of Object-Oriented Languages and Systems, pp. 71-80, July 2000.
[5] L.H. Etzkorn, S. Gholston, and W.E. Hughes, "A Semantic Entropy Metric," J. Software Maintenance: Research and Practice, vol. 14, no. 5, pp. 293-310, July/Aug. 2002.
[6] A. Marcus and D. Poshyvanyk, "The Conceptual Cohesion of Classes," Proc. 21st IEEE Int'l Conf. Software Maintenance, pp. 133-142, Sept. 2005.
[7] Bela Ujhazi, Rudolf Ferenc, D. Poshyvanyk, and Tibor Gyimothy, "New Conceptual Coupling and Cohesion Metrics for Object-Oriented Systems"
[8] Foltz, P. W., Kintsch, W. & Landauer, T. K. (1998), New Mexico State University. The measurement of textual coherence with Latent Semantic Analysis. Discourse Processes, , 25,2&3, 285-307.
[9] Cara Stein, Letha Etzkorn, S. Gholston, Philip Farrington, and Julie Fortune, "A Knowledge Based Cohesion Metric for Object Oriented Software" IEEE Journal of computer science, vol. 5, no 3, pp. 45-53