ISSN:2231-0711
www.ijcset.net

A Class Level Fault Prediction in ObjectOriented Systems: Cohesion Approach

M.Sudhir Kumar[1], S.V.AchutaRao[2], SyedAzar Ali[3], MohammedAfroze[4], Amjan.Shaik[5]

[1]*CSE, ECET, Patelguda, Hyderabad, Andhra Pradesh, India.*
[2]*CSE,IT,VCET,Nunna, Vijayawada, Andhra Pradesh, India*
[3]*IT, MJCET, GandipetX-Road, Hyderabada, Andhra Pradesh, India.*
[4]*Computer Science Department, King Saud University, Riyadh.*
[5]*Department of CSE,ECET,Patelguda, Hyderabad, Andhra Pradesh, India.*

Abstract-High cohesion is desirable property in software systems to achieve reusability and maintainability. Software metrics plays vital role in software development. Here we are measures for cohesion in ObjectOriented (OO) software reflect particular interpretations of cohesion and capture different aspects of it. In earlier approaches the cohesion was calculated from the structural information, for example method attributes and references. Now inConceptual Cohesion of Classes(CCC), we are calculating the unstructured information from the source code such as comments and identifiers. Unstructured information is embedded in the source code. To retrieve the unstructured information from the source code Latent Semantic Indexing is used. A case study on open source software systems is presented, which compares the new measure with an extensive set of existing metrics and uses them to construct models that predict software faults. In this paper we are achieving the high cohesion and we are predicting the fault in Object Oriented Systems.

Keywords: Cohesion, Reusability, Object Oriented.

I. INTRODUCTION

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 [1,3,5,6] etc.

Software cohesion can be defined as a measure of the degree to which elements of a module belong 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[1,2,3].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, reuse effort, prediction of software quality, fault prediction modularization of software and identification of reusable of components[5,6,24,27].

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 tradeoff 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[10,11,13]. 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 to capture different aspects of cohesion, only a few metrics address the conceptual and textual aspects of cohesion[12,15,18].

Propose a new measure for class cohesion, named the Conceptual Cohesion of Classes (CCC), 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[1,10,17,20].

CCC 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 [23,27,30,33].

Cohesion ultimately affects the comprehensibility of source code. For the source code to be easy to understand, it has to have a clear implementation logic (that is, design) and it has to be easy to read (that is, good language use). These two properties are captured by the structural and conceptual cohesion metrics, respectively.

The rest of the paper organizedis as follows. In section II, theBack Ground discussed, section III explores the Methodology, section IV Latent Semantic Indexing, section V shows the experimental results,section VI with conclusion and that followedby references.

II. BACK GROUND

Structural metrics are calculated from the source code such as references and data sharing between methods of a class belong together for cohesion.It define and measure relationships among the methods of a class based on the number of pairs of methods that share instance or class variables one way or another for cohesion. Therefore the result is Lackingof high cohesion[1,5,7,9,19,24,27,35].

III. METHODOLOGY

In this paper un-structural information is retrieved from the source code like comments and identifiers.Information is retrieved from the source code using Latent Semantic Indexing.With the help of CCC and existing metrics we are achieving the high cohesion and low coupling.We can predict the fault prediction using high cohesion. This work will be applicable in well compiled java program and it should have valid comments to measure the cohesion.

- Retrieving the structured information.
- Check the availability of structured information for source code.
- Apply the LCOM5 formula for structured information.
- Analyze about the comments i.e. unstructured information.
- Index Searching
- Apply the Conceptual similarity formula.
- Comparison

A. Modules and Description:

Module1:
To take the structured information like identifiers, (Example Variables).Invocation of declared methods and declared constructors. Here the Java program should be well compiled and it should be valid comments.

Module2:
To search the declared variables among all the classes. Because the main theme of the declaring class variable is, it should be used in all methods. So that the declared variables are found among all the methods.

Module3:
To apply the LCOM5 (Lack of cohesion in methods) formula. If the result is equal to one means, the class is less cohesive according to the structured information.

Module4:
To retrieve the index terms based on that comments which are present in all the methods. Comments are useful information according to the software engineer. In concept oriented analysis we are taking the comments. Based on the comments we are going to measure the class is cohesive or not.

Module5:
To check the index terms among the comments which are present in all the comments.

Module6:
To apply the conceptual similarity formula. Based on the result we can say the class is cohesive or less cohesive according to concept oriented.

Module7:
To compare the two results. Based on the results we can say that cohesion according to structure oriented and unstructured oriented.

B. Validations:

Module1
Input: Java program.
Result:Extracting the structured information from the class.

Module2
Input: Class variables and methods alone.
Result: Extracted terms are checked out among the all methods.

Module3
Input: Checked one.Apply the LCOM5 formula
Result: Cohesive or not.

Module4
Input:Java program.
Result:Extract the comments.

Module5
Input: Comments.
Result:Extract the index terms.

Module6
Input: Index terms.
Result: Checked one, Based on that Class is cohesive or not.

Module7
Input: LCOM5.
Result: Comparison.

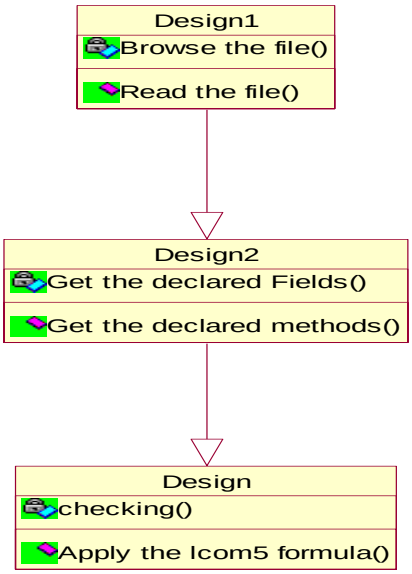

Figure:1 Class Diagram

IV. 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 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. The central concept of LSI is that the information about the contexts in which a particular word appears or does not appear provides a set of mutual constraints that determines the similarity of meaning of sets of words to each other.

LSI was originally developed in the context of IR as a way of overcoming problems with polysemy 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) . As a result, LSI offers a way of assessing semantic similarity between any two samples of text in an automatic unsupervised way.

LSI relies on an SVD of a matrix (word_context) derived from a corpus of natural text that pertains to knowledge in the particular domain of interest. According to the mathematical formulation of LSI, the term combinations that occur less frequently in the given document collection tend to be precluded from the LSI subspace. LSI does “noise reduction,” as less frequently co-occurring terms are less mutually related and therefore less sensible.Similarly, the most frequent terms are also eliminated from the analysis. The formalism behind SVD is rather complex and too lengthy to be presented here.

Once the documents are represented in the LSI subspace,the user can compute similarity measures between documents by the cosine between their corresponding vectors or by their length. These measures can be used for clustering similar documents together to identify “concepts” and “topics” in the corpus. This type of usage is typical for text analysis tasks.

Merits

1. We can predict the Cohesion.
2. We can predict the particular system is cohesive or not.

V. EMPIRICAL RESULTS

Figure:2 Cohesion measurement for class

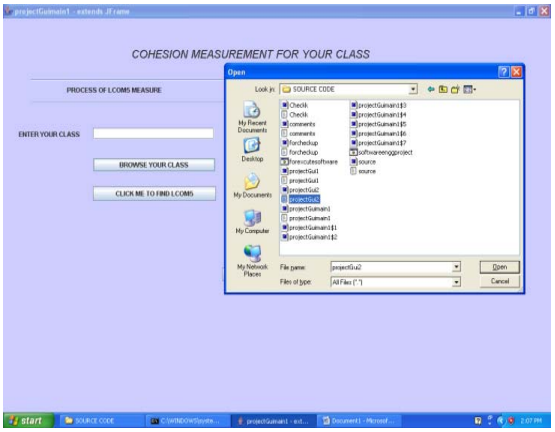

Figure:3 Executing the project as source code

Figure:4 Enter a specific class for Lcom5

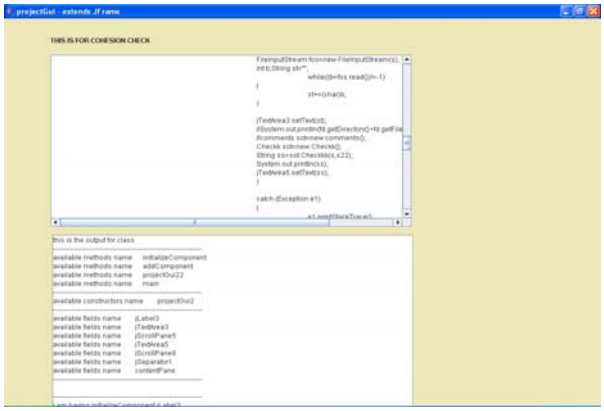

Figure:5 Measuring the cohesion

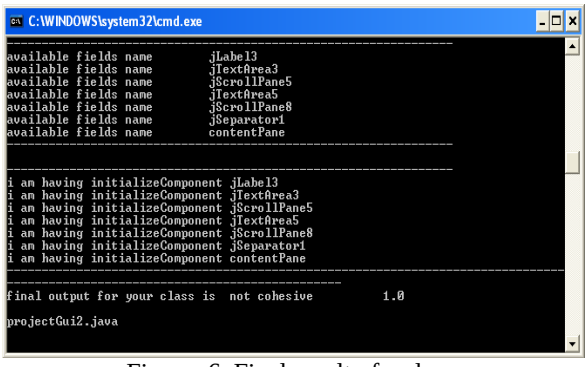

Figure: 6 Final result of a class

Figure: 7 Process of Lcom5 and CCC

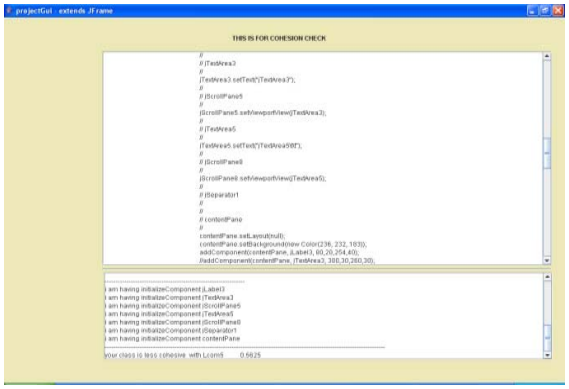

Figure:8 Comparing a class with Lcom5

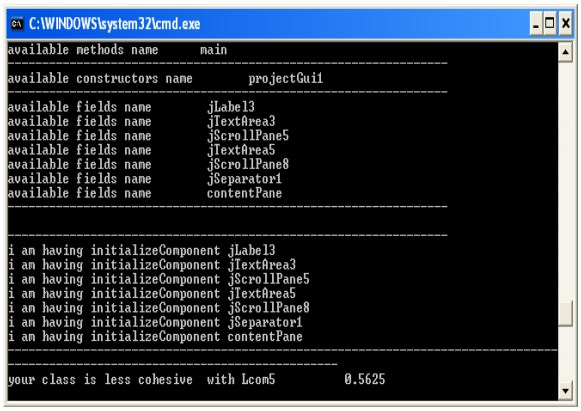

Figure: 9 Final result of a class for Lcom5

VI. CONCLUSION

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 beused 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 open source software applications 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 ofstructural metrics alone. Highly cohesive classes need to have a design that ensures a strong coupling among its methods and a coherent internal description.

ACKNOWLEDGEMENTS

Authors would be thankful and appreciate the R&D Cell, ECET for gathering the information and to prepare this paper.

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.

298 IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 34, NO. 2, MARCH/APRIL 2008

[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, pp. 751-761, 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," Proc. Symp. Software Reusability, pp. 259-262, 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] AmjanShaik, C. R. K. Reddy, BalaManda, Prakashini. C, Deepthi. K" An Empirical Validation of Object Oriented Design Metrics in Object Oriented Systems" International Journal of Emerging Trends in Engineering and Applied Sciences (IJETEAS) 1 (2): 216-224 (ISSN: 2141-7016).
[12] AmjanShaik, C. R. K. Reddy, BalaManda, Prakashini. C, Deepthi," Metrics for Object Oriented Design Software Systems: A Survey "International Journal of Emerging Trends in Engineering and Applied Sciences (IJETEAS) 1 (2): 190-198 (ISSN: 2141-7016).
[13] AmjanShaik, C. R. K. Reddy, BalaManda." Empirically Investigating the Effect Of Design Metrics On Fault Proneness in Object Oriented Systems" International Journal of Computer Science & Engineering Technology (IJCSET)
[14] R.S. Pressman; *"Software Engineering - A practitioner's approach"*, sixth edition, McGraw Hill, 2005.
[15] I. Sommerville; *"Software Engineering"*, 7th edition, Addison Wesley, 2004.
[16] El Emam, K.; Benlarbi, S.; Goel, N. and Rai, S. N; *" The confounding effect of class size on the validity of object-oriented metrics"*, IEEE Transactions on Software Engineering, 27(7) - 630-650, 2001.
[17] N. Fenton et al.; *"Software Metrics - A Rigorous and Practical Approach"*, International Thomson Computer Press, 1996.
[18] B. Henderson-Sellers; *"Object-Oriented Metrics, Measures of Complexity"*, Prentice Hall, 1996.
[19] Puneet Jai Kaur, AmadeepVerma and SimrandeepThapar; *"Software Quality Metrics for Object-oriented Environments"*, In proceedings of National Conference on Challenges & Opportunities in Information Technology, MandiGobindgarh, 23th March,2007.
[20] Chidamber, Shyam and Kemerer, Chris, "A Metrics Suite for Object Oriented Design", IEEE Transactions on Software Engineering, June, 1994, pp. 476-492.
[21] Lorenz, Mark and Jeff Kidd, *Object-Oriented Software Metrics*, Prentice-Hall, Englewood Cliffs, N.J., 1994.
[22] Tegarden, D., Sheetz, S., Monarchi, D., "Effectiveness of Traditional Software Metrics for Object Oriented Systems", Proceedings - 25th Hawaii International Conference on System Sciences, January, 1992, pp. 359-368.
[23] Williams, John D., "Metrics for Object Oriented Projects", Proceedings - Object ExpoEuro Conference, July, 1993, pp. 13-18.
[24] Booch, Grady, Object Oriented Analysis and Design with Applications, The Benjamin/Cummings Publishing Company, Inc., 1994.
[25] Jacobson, Ivar, Object Oriented Software Engineering, A Use Case Driven Approach, Addison-Wesley Publishing Company, 1993.
[26] Hudli, R., Hoskins, C., Hudli, A., "Software Metrics for Object-oriented Designs", IEEE, 1994.
[27] Lee, Y., Liang, B., Wang, F., "Some Complexity Metrics for Object-Oriented Programs Based on Information Flow", Proceedings - CompEuro, March, 1993, pp. 302-310.
[28] McCabe & Associates, McCabe Object-Oriented Tool User's Instructions, 1994.
[29] Set Laboratories, UX Metrics, 1994.
[30] Sharble, Robert, and Cohen, Samuel, "The Object-Oriented Brewery-A Comparison of Two OO Development Methods", Software Engineering Notes,Vol 18, No 2, April 1993,pp 60 -73.
[31] Shatnawi, R; A Quantitative Investigation of the Acceptable Risk Levels of Object-Oriented Metrics in Open-Source Systems, in IEEE Transactions on Software Engineering, Volume - 36 Issue - 2, On page(s) - 216 – 225, March-April 2010.
[32]http-//www.aivosto.com/project/help/pm-limits.html; visited the website in 13, dec,2010.
[33] Hans Erik Eriksson, Magnus Penker, Brain Lyons and David Fado; "UML 2 Tool kit", Willey publication, 2007.
[34] Albert Dieter Ritzhaupt , "Object-Oriented Design Metrics Using UML Class Diagrams " , 2nd CCEC Symposium 2004, April 8, 2004,UNF, Jacksonville, FL. USA.
[35] Alessandra Cau, GiulioConcas, Michele Marchesi "Extending Open BRR with Automated Metrics to Measure Object Oriented Open Source Project Success", May 19, 2006
[36] F.B. Abreu and R. Carapuca, "Candidate Metrics for Object-Oriented Software within a Taxonomy FraFramework," . System and Software,vol. 26, no. l, pp. 87-96, Jan. 1994
[37] J.Bansiya and C.G.Davis, "A Hierarchical Model for Validation of Object Oriented Design Quality Assessment", IEEE Transactions on Software Engineering, Vol.28, No 1, January, 2002.
[38] J M Bieman and B.-K. Kang, "Cohesion and Reuse in an Object-Oriented System," Proc ACM SIGSOFT Symp Software reusability, Seattle, Wash., pp. 259-262,1995.

ABOUT THE AUTHORS:

M.Sudhir Kumar is working as an Associate Professor, Department of CSE at Ellenki College of Engineering and Technology (ECET), Hyderabad, India. He has received B.Tech(CSE) from JNTU Kakinada and M.Tech(SE) from JNTUH Hyderabad. He has been presented and published 2 articles in International Conferences. His research interests are Software Engineering, Software Metrics and Information Security.

S.V. AchutaRao is working as a Professor and Head, Department of CSE and IT at VCET,Nunna,Vijayawada, India.He has received M.Tech. (Computer Science and Engineering) from JNTU, Kakinada, India. Presently, he is a Research Scholar of Rayalaseema University (RU), Kurnool, India. He has been Published 5 Research Papers in various International Journals. His main research interests are Bio-informatics, Software Metrics,Data Mining, Networking and Image Processing.

Syed Azar Aliis working as an Assistant Professor, Department of IT atMuffakhamJah College Of Engineering and Technology (MJCET) , GandipetX-Road, Hyderabad, India. He has received B.Tech(IT) and M.Tech(IT) from JNTUH Hyderabad. He has been presented number of technical papers in International and National Conferences. He published aResearch paper in International Journal. His mainresearch interests are Software Engineering, Software Metrics, Information Security and Image Processing.

Mohammed Afroze is working as aIT Trainer,Computer Science Department, King Saud University, Riyadh. He has received B.Tech(CSE) and M.Tech.(CSE) from JNTUH, Hyderabad. He has been presented Research and Technical Papers in National and International Conferences. His main researchinterests are Software Engineering , Software Metrics,Data Mining and Image Processing.

AmjanShaik is working as a Professor Department of Computer Science and Engineering at Ellenki College of Engineering and Technology (ECET), Hyderabad, India.He has received M.Tech. (Computer Science and Technology) from Andhra University.He hasbeen published and presented more than 30 technical and Research papers in International, National Conferences and International Journals. His main research interests are Software Engineering, Software Metrics and OOAD.