Concept Analysis for Class Cohesion

Padmaja Joshi* and Rushikesh K. Joshi
Department of Computer Science and Engineering
Indian Institute of Technology Bombay
Powai, Mumbai-400076, India.
Email: padmaja@cse.iitb.ac.in, rkj@cse.iitb.ac.in

Abstract

A concept lattice based approach for analysis of class cohesion is presented. The approach facilitates rapid identification of less cohesive classes. It also helps identify less cohesive methods, attributes and classes in one go. Further, the approach guides refactorings such as extract class, move method, localize attributes and remove unused attributes. The effectiveness of the technique is demonstrated through examples.

1 Introduction

Development-time cohesion analysis of individual classes can give an early indication of design flaws in classes. The approaches available to cohesion analysis are primarily metric based and concept lattice based. A number of object oriented cohesion metrics can be found in the literature. However, the metric based approach has a few drawbacks. Most cohesion metrics compute a single value for the entire class. Though the metrics may detect a less cohesive class, they do not help identify individual members contributing to lack of cohesion.

The proposed work bridges these gaps through a concept based approach. A developer can obtain an early feedback on the goodness of a class through visualization of a concept. The concept lattices generated with this approach are named as *Cohesion lattices* as they capture cohesiveness of a class and its members.

The rest of the paper is organized as follows. The next section discusses related work. Section 3 formulates the cohesion concept used in the lattice-based approach. Section 4 demonstrates applicability and validity of this technique with the help of examples and a case study.

*Presently with Center for Development of Advanced Computing, Mumbai, India.

2 Related Work

The notion of microscopic metrics for refactoring was brought out in our earlier work on coupling [7], in which, refactorings *Move method* and *Move field* were guided by two new microscopic metrics RMC and RIC. Distance [11] and Structural cohesion [9] are two examples of microscopic cohesion metrics that guide refactoring. Lucia et al. also take a metric based approach for guiding *Extract class* refactoring. However, metric based analysis may be cumbersome and complex as compared to the concept based visual approach demonstrated in this paper. Concept based cohesion analysis is aimed at providing cohesion information about a class at a single glance.

A few examples of the concept based approach for cohesion analysis can be found in the literature. Lindig and Snelting [8] use methods accessing attributes as a concept to find modular structures in legacy procedural code. The technique also identifies horizontally decomposable units in the code. Siff and Reps [10] use the same concept to identify modules from the C code. Their approach also considers attributes that are not accessed by functions to identify the modules.

The above concept that was used to identify modules in procedural code has later been used by Dekel [2, 3] to analyze understandability of classes in object oriented systems. He uses the concept of attributes defined and accessed by methods in a class. In this approach, the concept analysis is used to suggest a method sequence to be followed while reading a class for better understanding. Similar concept is also used by Sutton and Maletic [14] to extract UML models from C++ code.

Our approach uses the same concept as that of Dekel with the focus now on class cohesion and refactoring. Our approach helps identify class members contributing to lack of cohesion. Further, the approach suggests refactorings for the less cohesive classes and members. The resulting improvement in the cohesiveness of such classes is demonstrated with the help of examples and a case study.

1534-5351/09 $25.00 © 2009 IEEE
DOI 10.1109/CSMR.2009.54

3 The Approach

The terminology followed in this paper is from Ganter et al. [6]. The context and the concept on which the cohesion lattices for classes are formulated are outlined below.

Let D_c be the attributes defined in class c, I_c be the inherited attributes that are referred in class c, M_c be the methods defined in class c, and A_c be the attribute set $D_c \cup I_c$. A relation *Access* between sets A_c and M_c is defined as $Access = \{< a, m > | a \in A_c, m \in M_c, m$ refers to a directly or indirectly$\}$. The triple $(A_c, M_c, Access)$ is the *formal context* of the cohesion lattice for class c with attribute set A_c as the *extent set* and method set M_c as the *intent set*. Pair $(A_i \subseteq A_c, M_i \subseteq M_c)$ is considered as a *formal concept* of the context $(A_c, M_c, Access)$ iff ${A_i}' = M_i$ and ${M_i}' = A_i$, where, ${A_i}' = \{m \in M_c | (a, m) \in Access \ \forall a \in A_c\}$ and, ${M_i}' = \{a \in A_c | (a, m) \in Access \ \forall m \in M_c\}$. The cohesion lattices are formulated in terms of these formal concepts.

Let T be the set of all concepts in concept lattice L. Concepts $Infimum$ and $Supremum$ are identified as $Infimum_L = (\bigcap_{t \in T} A_t, M_c)$, and $Supremum_L = (A_c, \bigcap_{t \in T} M_t)$. A concept $(A_i, M_i) \in L$ is called as a filled concept when its extent and intent are non-empty i.e.$(A_i \neq \emptyset) \wedge (M_i \neq \emptyset)$. For some classes, infimum and supremum concepts may not form filled concepts. Two concepts are cohesive when there is a path between them that traverses only through *filled concepts*.

3.1 Cohesiveness and Cohesion Lattices

Cohesion lattices provide a lot of information not only about the class cohesion but also about the contribution of class members to class cohesion. Elements associated with the same concept show very high cohesion with each other as all the methods associated with the concept use all the attributes associated with it. Elements associated with two concepts are cohesive if the two concepts satisfy subconcept-superconcept relationship. Width of the lattice depicts reduction in cohesion as at least two concept are there which do not share all its properties with the other concept.

Unconnected nodes in a lattice signify the elements that are not related with each other. When a class follows such lattice, the class is uncohesive. The least cohesive attributes are accumulated at the filled supremum. If only a single method is associated with this node, then the attributes can be moved to that method. Pure functions are associated only with the unfilled infimum (infimum with $\emptyset$ objects). These methods may be extracted as a separate class, as these methods do not use any of the instance variables and hence are not very cohesive with the remaining elements. Similar to pure functions, unused attributes are associated with the unfilled supremum. These attributes need to be justified else can be removed.

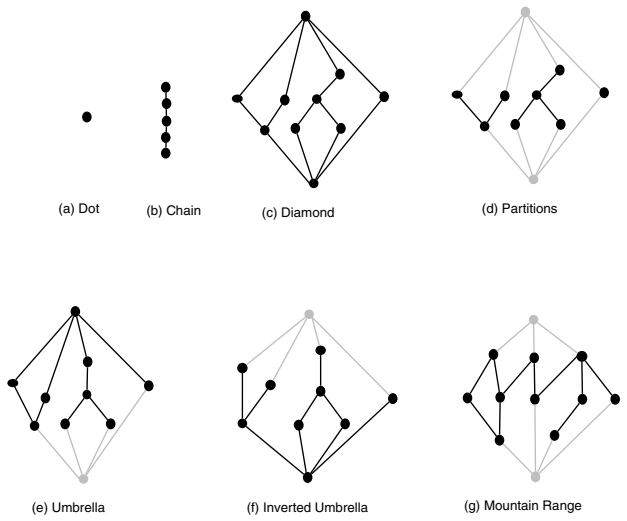

Figure 1. Some Cohesion Lattice Structures

Commonly observed cohesion lattice structures are identified in Figure 1. The detailed formal characterization of these structures is omitted due to lack of space.

A *Dot* lattice is the one in which the supremum is the same as that of the infimum. It represents a most cohesive classes. A *Chain* lattice is a highly cohesive lattice in which the concepts form a chain structure. A *Diamond* lattice shows some reduction in cohesion as some of the concepts are spread out widthwise. However, it represents a cohesive class. An *Umbrella* lattice has unfilled infimum and filled supremum. This structure stands cohesive mainly due to the methods in the supremum. An *Inverted Umbrella* lattice has filled infimum and unfilled supremum. It is a cohesive lattice since all its methods share at least one attribute. A *Mountain Range* lattice has both infimum and supremum unfilled, but all its concepts stay connected. It represents a less cohesive class that needs inspection. A mountain range with disconnected components represents a *Partitions* lattice that attracts *Extract class* refactoring.

4 Applying the Technique to Refactoring

The concept based technique can be applied to classes irrespective of their sizes. Though for big classes, the lattice may become complex, the identification of less cohesive instance variables and methods can be easily done as lattice drawing tools such as *Concept Explorer* [12] can provide features such as highlighting of connectivity.

Example 1: A Java-based spell-checker for Marathi, an Indian Language [4] is taken as a case. It consists of 15

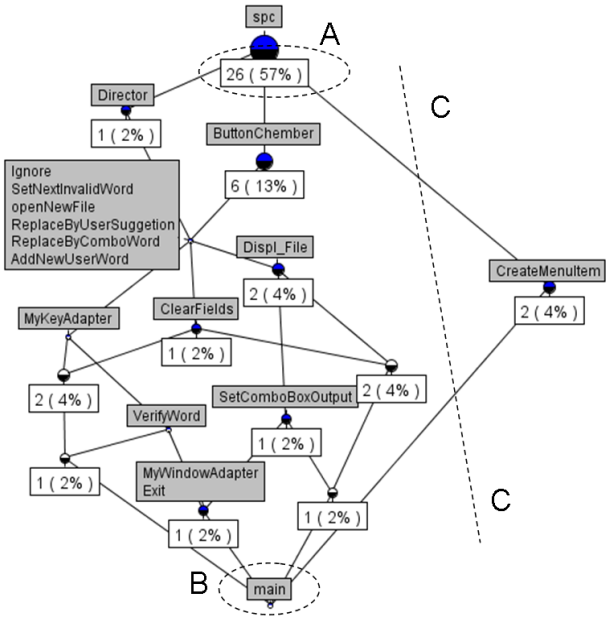

Figure 2. *Umbrella* **Lattice for class** *spc*

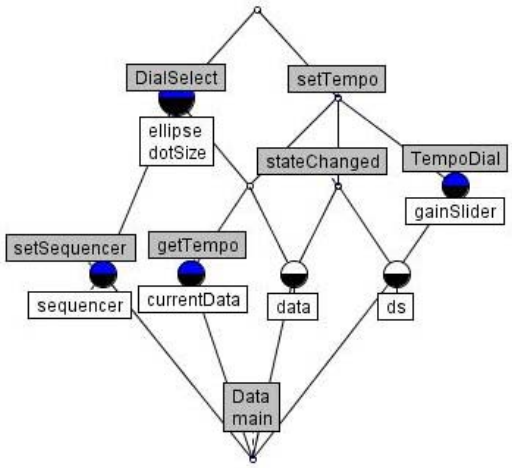

Figure 3. *Mountain Ranges* **for class** *TempoDial*

Java classes including three derived classes and about 135 methods. Cohesion lattice for class *spc* from this application is shown in Figure 2. The lattice shows Umbrella structure with the constructor in its supremum. The structure suggests the following refactorings: (A) A bunch of instance variables in the supremum can be localized to the sole method *spc* in the supremum. (B) Method *main()* at the infimum does not use any of the instance variables and contains only testing code, which can be extracted as a separate class. (C) Removal of the constructor at supremum converts Umbrella into partitions and hence subsequently method *createMenuItem()* can be extracted in a separate class. This extraction improves the re-usability of the method and the class cohesion.

Example-2: Figure 3 shows a Mountain Range for class *TempoDial* from the Java sound demo classes [13]. It can be seen from the figure that the class is a well designed class. However, it contains two methods *Data* and *main* in the unfilled infimum. Extraction of this testing code as a separate class makes *TempoDial* a good cohesive class.

The above application contains all lattice structures except Diamond and Partitions. Eight classes were refactored based on the technique proposed. Table 1 shows the refactorings applied to the application. The refactorings were further validated with the help of various cohesion and coupling metrics. Improvement was observed with both kinds of metrics.

Table 1. Summary of Refactoring Applied

Refactoring Applied	No.of Classes
Uncohesive attributes (Delete or localize)	2
Uncohesive Methods (Extract as a class)	5
Extract class due to partition	1
No refactoring needed	7

4.1 Metric Based Validation

The approach was validated with the help of many coupling and cohesion metrics. TCC [1] and SCOM [5] based validation is reported ahead. TCC [1] captures cohesion in terms of the ratio of the number TCC [1] captures cohesion in terms of the ratio of the number of connected method pairs to the total number of method pairs. Pure function extraction reduces total number of pairs, improving TCC. Similarly, when a partition is extracted as a separate class, a considerable improvement is seen. TCC values for six classes from the case study are given in Figure 4.

SCOM [5] captures the negative contribution of uncohesive attributes. In SCOM, the relatedness between method pairs is averaged over the total number of attributes and the total number of method pairs. The improvements due to refactorings such as localization of attributes, removal of unused attributes, extraction of pure functions as a separate class and extract class get captured through SCOM due to reduction in the denominator of SCOM ratio. The results for five classes are shown in Table 2. Improvement in class cohesion is clearly visible in Figure 4 and Table 2.

Lattice formation needs almost the same information that a typical metric computation needs. In metric based ap-

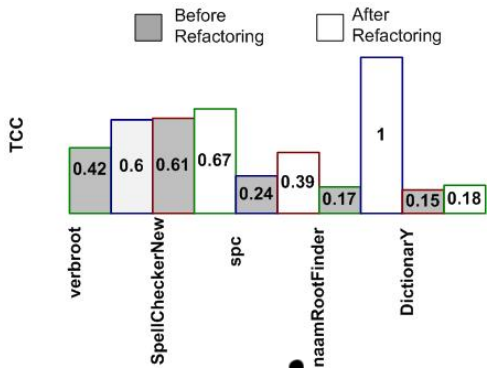

Figure 4. TCC based Validation

Table 2. SCOM based Validation

Refactoring	Number	DictionarY	d-node	scn	naamRootFinder
Before	0.167	0.04	0.4	0.02	0.03
After	1.0	0.07	0.56	0.05	0.043

proach, various metric values for every class member need to be checked against certain thresholds to identify members for refactoring. Whereas, the concept based approach provides rapid visual identification of uncohesive classes and members.

5 Conclusions

A concept based technique was introduced for class cohesion analysis and refactoring. With this approach, the cohesiveness of a class can be visualized from the cohesion lattice structure of the class. Seven cohesion lattice structures of varying degree of cohesiveness are identified. A cohesion lattice also brings out the contribution of each member to class cohesion. Identification of least cohesive attributes and pure functions gets simplified with the concept based approach as the least cohesive attributes are always present in the supremum, and the pure functions can be found at the infimum. Such attributes can be localized or can be deleted and the pure functions can be extracted as a separate class. Unconnected members also should be extracted as a separate class. The concept based cohesion analysis gives a quick visual overview of class cohesion and helps guiding refactorings such as localization of attributes, removal of instance variables and class extraction.

References

[1] J. M. Bieman and B.-K. Kang. Cohesion and reuse in an object-oriented system. In *Proceedings of ACM symposium for software reusability*, pages 259–262, 1995.

[2] U. Dekel. Application of concept lattices to code inspection and review. In *The Insreli Workshop on Programming Languages and developmeny Environment*, July 2002.

[3] U. Dekel and Y. Gil. Revealing class structure with concept lattices. In *IEEE Proceedings of 10th Working Conference on Reverse Engineering (WCRE)*, November 2003.

[4] V. Dixit, S. Dethe, and R. K. Joshi. Morphology-based spellchecking for marathi, an indian language. In *Proceedings of Language and Technology Conference*, Poznan, Poland, April 2005.

[5] L. Fernández and R. Pena. A sensitive metric of class cohesion. *International Journal Information Theories and Applications*, 13:82–91, 2006.

[6] B. Ganter, G. Stumme, and R. Wille. *Formal Concept Analysis Foundations and Applications*. Springer, 1998.

[7] P. Joshi and R. K. Joshi. Microscopic coupling metrics for refactoring. In *IEEE Proceedings of 10th European conference on software maintenance and reengineering (CSMR)*, pages 145–152, Bari, Italy, March 2006.

[8] C. Lindig and G. Snelting. Accessing modular structure of legacy code based on mathematical concept analysis. In *Proceedings of the 19th International Conference on Software Engineering (ICSE)*, pages 349–359, May 1997.

[9] A. D. Lucia, R. Oliveto, and L. Vorraro. Using structural and semantic metrics to improve class cohesion. In *Proceedings of International Conference on Software Maintenance (ICSM)*, pages 27–36, 2008.

[10] M. Siff and T. Reps. Identifying modules via concept analysis. *IEEE Transactions on Software Engineering*, 25:749–766, NOv/Dec 1999.

[11] F. Simon, F. Steinbruckner, and C. Lewerentz. Metrics based refactoring. In *International Proceedings of Fifth European Conference on Software Maintenance and Reengineering (CSMR)*, pages 30–38, 2001.

[12] SourceForge. Concept explorer. Website, 1999-2009. http://sourceforge.net/projects/conexp/.

[13] Sun Microsystems. Java sound api: Java sound demo. website, 1994-2009. http://java.sun.com/products/java-media/sound/samples/JavaSoundDemo//.

[14] A. Sutton and J. Maletic. Recovering UML class models from c++: A detailed explanation. *Information and Software Technology*, 49:212–229, March 2007.