Abstract— Object oriented programming has taken a major role in the development of large, complex systems. To ensure the software quality of these systems there is need for object oriented metrics to predict and control the quality of these systems. The internal properties which has impact on quality of these large, complex object oriented systems are cohesion, coupling, inheritance. A highly cohesive module is one whose elements have a close relationship among them in order to provide a single basic functionality of the module. On the contrary, a low cohesive module has elements that have less or no relation with each other leads to poor structure which makes it difficult to maintain. The structure of software systems is the subject of many changes during the systems lifecycle. Improper implementations of these changes implies structure degradation that leads to costly maintenance. When developing object-oriented classes, it is difficult to determine how to best reallocate the members of large, complex classes to create smaller, more cohesive ones. Clustering techniques can provide guidance on how to solve this allocation problem. A continuous improvement of the software systems structure can be made using refactoring, that assures a reliable and easy to maintain software structure.

Keywords— Cohesion, Coupling, Clustering, Metric, Refactoring

I. INTRODUCTION

The structure of a software system has a major impact on the maintainability of the system. That is why continuous restructurings of the code are needed, otherwise the system becomes difficult to maintain, change and becomes unreliable. Code maintenance is expensive. A common maintenance problem in case of object-oriented systems is the presence of large, complex classes with many unrelated methods and attributes. This paper describes how the members of large classes can be suggested for refactoring using clustering techniques. Using the outputs of the clustering process, programmers can refactor their large classes and improve their software so as to simplify the maintenance and improve reliability.

Object oriented metrics are needed to monitor the special traits of object oriented system like data abstraction, encapsulation, inheritance. Cohesion, coupling are the internal characteristic which has an huge impact on the quality of object oriented system. Highly cohesive class is easy to maintain and less prone to error. Cohesion indicates the degree of binding between the elements of class as well as the functional strength of class. Highly cohesive class provides a single well stated functionality.

Cohesion in OO refers to the interactions among the attributes and methods in a class. Three possible types of interactions among the elements that contribute to class cohesion are method-method, method-attribute, and attribute-attribute interactions[3]. When two methods access a common attribute it contributes to a method-method interaction. When a method accesses an attribute it falls under the method-attribute category. When two attributes are accessed by the same method it results in an attribute-attribute interaction. A week cohesion means unrelated responsibilities/functions imply that the module will have reasons to change in the future.

Coupling indicates the interdependencies among software modules. A software design should have low coupling, that is the collaboration between modules should be as low as possible.

Fowler (Fowler et al. 1999) defines refactoring as “a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior”, and identifies a large class as being one of the class with “bad smells” in software that indicate likely problems. It is a disciplined way to improve the design of the code that minimizes the chances of introducing bugs. The refactoring process consists of the following distinct activities: Identify places where software should be refactored (known as bad smells) and determine which refactorings should be applied. Guarantee that the applied refactoring preserves behaviour. Apply the refactoring. Assess the effect of the refactoring on quality characteristics of the software.
Clustering is a data mining process used for partitioning a set of data (or objects) in a set of meaningful sub classes, called clusters, being considered the most important unsupervised learning problem. The resulting subsets or groups, distinct and non-empty, are to be built so that the objects within each cluster are closely related to one another than objects assigned to different clusters. Important to the clustering process is the notion of degree of similarity (or dissimilarity) between the objects.

Let \( O = \{O_1, O_2, \ldots, O_n\} \) be the set of objects to be clustered. The measure used for discriminating objects can be any metric or semi-metric function \( d : O \times O \rightarrow R \). The distance expresses the dissimilarity between objects.

In this paper a precise cohesion concept link strength is used as a basis for clustering algorithm that would help developers to identify the appropriate refactorings in a software system. The proposed approach takes an existing software and reassembles it using clustering, in order to obtain a better design, suggesting the needed refactorings. Applying the proposed refactorings remains the decision of the software engineer. The main contribution of this paper is to improve the approach by using accurate cohesion metric in clustering algorithm for identifying refactorings opportunities. The objective of this paper is to use pattern recognition techniques to help designers identify how their code can be refactored, at the class level, in order to maximize the cohesion.

II. RELATED WORK

There are various approaches in the field of refactoring. In [10], a search based approach for refactoring software systems structure is proposed. The authors use an evolutionary algorithm for identifying refactorings that improve the system structure. An approach for restructuring programs written in Java starting from a catalog of bad smells is introduced in [2]. Based on some elementary metrics, the approach in [4] aids the user in deciding what kind of refactoring should be applied.

Clustering techniques have already been applied for program restructuring. In [8] a clustering based approach for program restructuring at the functional level is presented. Lot of work has been done with respect to remodularising or partitioning or clustering large software modules. Some of them are [9], [10]. In all of the above works remodularisation of software modules in a higher level (like package or file level) is proposed.

III. MOTIVATION

As in object oriented systems large classes with unrelated elements may result during initial stage due to deviation from design principles or later during updation to meet new requirements. These classes with low cohesion are fault prone.

Refactoring is a challenging and time consuming task that requires efforts to identify and apply. The process of refactoring can be automated and supported by tool. However the challenging task is to identify the code that needs to be refactored. While there exist several cohesion metrics which captures the cohesiveness of the class most of the metrics have drawbacks with regard to their discriminative power. Since LCOM1 and LCOM2 were the first cohesion metrics proposed, they fall short on many accounts. The values in case of LCOM1 and LCOM2 can grow exponentially which makes it to difficult to compare.

In case of LCOM3 and LCOM4 both counts the number connected components but both fail in distinguishing between similar and identical classes. In case of LCC and TCC highest cohesion value is assigned for a method pair even if they have only one attribute in common.

The cohesion concept Link strength which is based on average attribute usage of class is accurate in terms of discriminative property of cohesion metric.

A. Average attribute usage

The Average Attribute Usage (AAU) computes the average number of attributes used by each method of the class. We consider only public, non-inherited methods of a class. Let \( c \) be the class in consideration. \( AR(m) \) is attributes referenced by method \( m \). \( M_{pub}(c) \) is set of public methods of class \( c \). \( M_{i}(c) \) is set of non-inherited methods of class \( c \).
B. Link strength

The Link Strength (LS) is based on the AAU. Link Strength for a pair of methods $m_1$ and $m_2$ is given by

$$AAU = \frac{\sum AR(m)}{|M_f(c) \cap M_{pub}(c)|}$$ (1)

$$LS_{m_1m_2} = \begin{cases} \frac{|AR(m_1) \cap AR(m_2)|}{AAU}, & \text{if } |AR(m_1) \cap AR(m_2)| \leq AAU \\ 1, & \text{if } |AR(m_1) \cap AR(m_2)| > AAU \end{cases}$$ (2)

LCOM4 considers the methods as nodes of a graph. Two nodes are connected with an edge if the methods have at least one attribute in common. Consider the example of class 5 in Figure 1. The class consists of 6 attributes and 3 methods. Graph shows single connected component so value is 1.

Observing the class we find that method $m_1$ and method2 share a single attribute $a_2$. Method $m_2$ and Method $m_3$ share two attributes $a_3$ and $a_4$. Method pairs $m_1$ and $m_3$ share three attributes $a_1$, $a_5$, $a_6$. So the linkage between $m_1$ and $m_2$ is not the same as that of $m_2$ and $m_3$ which in turn is different from the link between $m_1$ and $m_3$. The Link Strength metric is specifically meant to capture this difference in the strength of the links between methods. The graph for LCOM4 metric has been modified to include link strength in Figure 3. It is clear that though all three methods are linked, the level or strength of their links is different and this is captured in LS.

IV. CLUSTERING FOR REFACTORING IDENTIFICATION

Clustering helps to handle complexity of classes and also helps in understanding the underlying structure of the system which is very critical for refactoring.
A. Clustering

To cluster a set of entities into groups the features of these entities must be extracted. Based on the feature values, the entities are organized into groups. For refactoring at class level, methods are chosen as entities. This is because methods are the basic computational elements of classes. Feature set is characteristics of the entities to be evaluated. The feature are used to calculate how close two entities are. Entities are more similar if they share more common features. Therefore data members used as feature for entities.

Distance function is a function that measures the distance between the entities based on their feature set. Distance function should emphasize cohesion. Part of this parameterization involves the representation of the clusters, that is how one defines the feature set of the cluster and how the distance function takes those into account when computing the distance between groups or between groups and individual entities. As methods are entities which are grouped if the distance which indicates the link strength between the entity and already formed cluster is below threshold then new cluster is formed or else entity is added to the cluster this procedure repeats until no more changes are possible. Domain knowledge that should be considered by clustering algorithms to improve their ability to aid in refactoring large classes. Assign each entity (class member) to a single cluster. Repeat merging while the specified threshold value is not reached. Display the outcome of the algorithm as cluster tree.

Clustering for refactoring identification consists of following steps:

Preprocessing and parsing – The source code is parsed, and a list of entities and feature is extracted.

Cohesion computation- Link strength between every pair of methods which indicates the cohesion is computed.

Data extraction - The existing software system is analyzed in order to extract from it the relevant entities: classes, methods, attributes and the existing relationships between them.

Grouping - The set of entities extracted at the previous step are re-grouped in clusters using a clustering algorithm. The goal of this step is to obtain an improved structure of the existing software system.

Refactorings extraction - The formed cluster tree is observed to identify the code that can be suggested for refactoring.

B. Refactoring extraction

The refactorings which is able to identify are:

1) Move Method refactoring.

It moves a method m of a class C to another class C’ that uses the method most; the method m of class C should be turned into a simple delegation, or it should be removed completely.
The bad smell motivating this refactoring is that a method uses or is used by more features of another class than the class in which it is defined.

2) Inline Class refactoring

It moves all members of a class C into another class C’ and deletes the old class. The bad smell motivating this refactoring is that a class is not doing very much.

This refactoring is identified by decreasing number of elements in the partition.

3) Extract Class refactoring.

Creates a new class C and move some cohesive attributes and methods into the new class. The bad smell motivating this refactoring is that one class offers too much functionality that should be provided by at least two classes.

This refactoring is identified by increasing the number of method pair with link strength below threshold.

V. CONCLUSION AND FUTURE ENHANCEMENT

In this paper, an approach was proposed for identifying low cohesive classes and clusters of concepts in low cohesive classes potential for class level refactoring. The accurate cohesive value obtained based on concepts like average attribute use has used for suggesting classes for refactoring. The cumulative effect of several simple refactoring steps and the tool support for their automated application has made the refactoring process a widely accepted technique for improving software design. Our goal is to make object-oriented software easier to maintain by converting large, noncohesive classes into smaller, more cohesive ones. Because programmers will make the final judgment about how classes will be organized, it is important that they see how a particular recommendation came about. The Refactoring improves the quality and simplifies the maintenance.

Further work can be done in the following directions: As in large classes there will enormous number of the refactoring suggestion ranking the refactoring may help the designer to consider them.

REFERENCES

[12] www.antlr.org