Research Paper Engineering

Impact of Software Evolution in Clustering

CHRISTABEL WILLIAMS | POST-GRADUATE STUDENT, DEPARTMENT OF CSE, KARUNYA UNIVERSITY

ABSTRACT

Clustering is carried out for maintaining large amount of data. It is concerned about grouping of similar objects. The objects that are closely related to each other are grouped together. It is an unsupervised learning technique. By doing so it eases the work of maintaining bulk data. Clustering has many applications in various fields. It helps in reverse engineering, software maintenance and for re-building systems. It aims at breaking a larger problem into small pieces of understanding elements. So far many clustering methods were carried out to achieve better clustering result. It naturally requires different techniques to the classification and association learning methods. This paper aims at including the evolutionary data for carrying out clustering. Evolutionary data is nothing but when two classes are often changed together by a single transaction then the two classes are said to be evolutionarily dependent. By considering these data into account better results can be achieved.

KEYWORDS clustering, evolutionary, classification

1. INTRODUCTION

The field of reverse engineering is mainly focus on the process of discovering the structure of existing software system in an effort to understand, re-document and if possible modify those systems. Software clustering is an important discipline in reverse engineering and software maintenance. It deals with the automatic unsupervised grouping of software artifacts like functions, classes or files into high-level structures like packages, components, or sub-systems based on the similarity of the artifacts. Software clustering techniques are commonly employed to aid reverse engineering process. Their goal is to break the problem of understanding a large piece of software into smaller, easier to manage sub-problems, thus reducing the amount of effort involved in recovering the structure of a given software.

Software evolution is the term used to the process of developing software initially, then repeatedly updating it for various reasons. Recently software engineers have become aware of software evolution as an important and largely unused data source to enhance the software development and maintenance process. Information from the evolution of software projects has been used across many applications. It helps the project managers to control the development process, software architects to detect design flaws, developers to find related files or hidden dependencies and quality controllers to identify bugs.

1.1 PROBLEM STATEMENT

Methods that integrate software clustering and software evolution, such as those discussed above, seem promising. On the one hand, software clustering based on structural data might be improved by integrating evolutionary data sources and, on the other hand, it appears to be possible to cluster software only by using evolutionary data. But despite these positive results, there are some aspects that are not covered sufficiently yet: The clustering-centered approaches first of all present new clustering techniques and only secondarily employ evolutionary data. The evolution-centered approaches just show that clustering based on evolutionary information is working to some degree, but are not contrasted to approaches based on structural data.

1.2 OBJECTIVE

Methods that use both software clustering and software evolution seem promising. Software clustering based on structural data might be improved by integrating evolutionary data sources. The clustering-centered approaches present new clustering techniques and only secondarily employ evolutionary data. The evolution-centered approaches just show that clustering based on evolutionary information is working to some degree, but are not contrasted to approaches based on structural data. The goal is to overcome these shortcomings by directly comparing structural and evolutionary data sources to each other.

1.3 PREVIOUS WORKS

Software clustering algorithms rely on characteristic information about the software artifacts to compute a pairwise similarity measure and to finally yield a reasonable clustering. Common clustering approaches retrieve this information directly from the static source code in form of structural dependencies based on, for example, method invocations and variable references among methods Andritsos et al. (2005) or inheritance, aggregation, and method invocations among classes Mitchell et al. (2007). Some approaches try to improve the clustering by taking dynamic code dependencies recorded during the program execution into consideration. Other approaches use the source code only indirectly by analyzing variable names and comments. Although there exists such a variety of data sources, only few approaches integrate several of them into their clustering technique.

1.4 PROPOSED METHOD

An established clustering algorithm is used to recover the architectures of the projects taken into consideration. Initially java classes are grouped into packages Beck et al. (2010). Classes are the elementary units in the design process of object oriented software systems. Their organization into packages reflects the architecture of a software system. Here along with the structural code dependencies the evolutionary dependencies are also considered and clustering is carried out using agglomerative algorithm.

2. EXPLANATION OF THE PROPOSED WORK

The following works should be carried out .it starts with selecting the data sources and extracting the relevant details.

2.1 DATA SOURCES

The data sources may be either structural or evolutionary. They are described in detail. A directed graph, where nodes represent classes and edges represent dependencies, models the asymmetric dependency information.

2.1.1 STRUCTURAL DEPENDENCIES

Structural static source code dependencies are the most wide-

ly used data source for software clustering. All types of structural class dependencies are incorporated namely inheritance, aggregation and usage. Beck et al. (2012) Let C(S) be the set of classes of a software system, we define the four directed relations on classes :

$$(c1, c2) \varepsilon E_{CIG} \Leftrightarrow c1 \text{ extends } c2 \quad (2.1)$$
$$(c1, c2) \varepsilon E_{CAG} \Leftrightarrow c1 \text{ aggregates } c2 \quad (2.2)$$
$$(c1, c2) \varepsilon E_{CUG} \Leftrightarrow c1 \text{ uses } c2 \quad (2.3)$$
$$E_{SCDG} := E_{CIG} \cup E_{CAG} \cup E_{CUG} \quad (2.4)$$

The directed graphs $G_x := (C(S), E_x)$ are called Class Inheritance Graph(CIG) given in (2.1), Class Aggregation Graph (CAG) given in (2.2), Class Usage Graph (CUG) given in (2.3) and Structural Class Dependency Graph (SCDG) given in (2.4).

2.1.2 EVOLUTIONARY DEPENDENCIES

The evolution of a software project is documented by the changes applied to its source files in the course of development. These changes are stored in a revision control system such as CVS or SVN. Class A depends on Class B by evolution if Class A has often been changed together with Class B. The support value of a dependency counts how often the two software artifacts were changed together. Additionally, the confidence value of a dependency relates the support to the total number of changes applied to one of the artifacts. Beck et al . (2012)

Let c1, c2 ε C(S) be two classes and $\{T_i\}^{l}_{i=1}$ sequence of transactions.

$$Supp(c_1, c_2) := |\{T_i : c_1, c_2 \varepsilon T_i\}| \quad (2.5)$$

is called support of the evolutionary dependency of class c1 to class c2.If c1 is element of at least one transaction Ti, the confidence of the evolutionary dependency of class c1 to class c2 is defined as

$$Conf(c_1, c_2) := \quad (2.6)$$
$$\text{Otherwise, } Conf(c_1, c_2) := 0$$

In the definition of Conf, the value of $Supp(c_1; c_1)$ given in (2.5) represents the total number of transactions c1 is part of. Thus, the value of Conf reaches its maximum 1 if c_2 is changed whenever c_1 is changed.Otherwise both the classes have been part of the same transactions. This is the basic idea behind evolutionary dependencies.Interpreting evolutionary dependencies is based on assumption that transactions implicitly group dependent files together.

2.2 EXPLANATION

Initially ten java real time projects are considered. From those files structural and evolutionary dependencies are extracted. Then the dependencies between them are tabulated. By using agglomerative clustering algorithm , clustering is carried out.

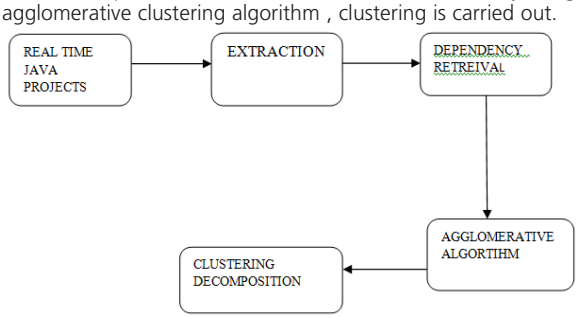

Fig 2.1. Block Diagram

2.3 CLUSTERING ALGORITHM

Clustering is carried out using agglomerative clustering algorithm. This is a "bottom up" approach: each observation starts in its own cluster, and pairs of clusters are merged as one moves up the hierarchy. Ying Zhao and George Karypis [3] defines certain clustering criterion that can be used to determine which clusters to merge at each step. For example in an n document dataset and the solution has been obtained after m merging steps. The solution includes n-1 clusters as one cluster, as one cluster will be removed in each step. So accordingly the next pair to be merged will be selected which will lead to a n-2 solution that optimizes the selected clustering criterion. That is one of the (n-1)*(n-2)/2 pairs of merges will be evaluated and the particular clustering criterion with the minimum value is selected. The criterion function will be locally optimized in that particular stage of the algorithm. Since agglomerative algorithm has computationally expensive steps and its complexity cannot be reduced for any particular clustering criterion it is not effective.

2.5 EVALUATION METHOD

To decide which data source performs best, we finally have to measure the quality of the clustering result. But evaluating software clustering results is challenging. There does not exist an optimal solution to the clustering problem due to different paradigms for designing software decompositions and individual preferences of developers. Hence, every evaluation method will be a heuristic. In this section we motivate our evaluation approach and discuss our design decisions.

CONCLUSION

A clustering based on evolutionary dependencies is only successful if substantial evolutionary data is available. Evolutionary dependencies often do not cover the set of classes sufficiently. The main advantages over a purely structural based clustering are that also non-source files can be considered and that the approach works independent of the programming language. An integration of the two data sources unites the advantages of the approaches at the cost of a more complex data acquisition. The clustering approach is still able to handle non-source files and non-parsed source files.

In this work are four modules which are class identification, structural dependency and evolutionary dependency extraction, module dependency graph and clustering. The result of the implementation shows that by including evolutionary dependencies, better clustering quality can be achieved. In the future Genetic algorithm can be used to carry out clustering since it overcomes the drawbacks of the agglomerative algorithm and hill climbing methods can be used in selection if the desired chromosomes from the available population and better results can be achieved.

REFERENCES

Andritsos P &Tzerpos V (2005) Information-theoretic software clustering. IEEE Transactions | on Software Engineering 31(2):150-165 | | Beck F, Diehl S (2010a) Evaluating the impact of software evolution on software clustering. | In:WCRE '10: Proceedings of the 17thWorking Conference on Reverse Engineering, | IEEE Computer Society, pp 99-108 | | Beck F & Diehl S (2012) On the impact of software evolution on software clustering– | Empirical Software Engineering October 2013, Volume 18, Issue 5, pp 970-1004 | | Mitchell BS, Mancoridis S (2007) On the evaluation of the Bunch search-based software | modularization algorithm. Soft Computing 12(1):77-93 | | Ying Zhao & George Karypis Comparison of Agglomerative and Partitional Document | Clustering Algorithms. Department of Computer Science, University of Minnesota, | Minneapolis, MN 55455 |