Using Structural and Semantic Information to Support Software Refactoring

Gabriele Bavota
School of Science, University of Salerno
Fisciano (SA), Italy
gbavota@unisa.it

***Abstract*—In the software life cycle the internal structure of the system undergoes continuous modifications. These changes push away the source code from its original design, often reducing its quality. In such cases refactoring techniques can be applied to improve the design quality of the system. Approaches existing in literature mainly exploit structural relationships present in the source code, e.g., method calls, to support the software engineer in identifying refactoring solutions. However, also semantic information is embedded in the source code by the developers, e.g., the terms used in the comments. This research investigates about the usefulness of combining structural and semantic information to support software refactoring.**

***Keywords*-refactoring; semantic information**

I. Research Problem and Hypothesis

During software evolution change is the rule rather than the exception [1]. In consequence of changes often software quality decreases, resulting in more and more difficulties in changing existing software [2]. In such cases a refactoring of the system is recommended since several empirical studies provided evidence that low design quality is generally associated with lower productivity, greater rework, and more significant efforts for developers [3]. Applying the correct refactoring operations[1] it is possible to improve different quality aspects of a system. As examples, class cohesion in Object-Oriented (OO) systems can be increased performing extract class refactoring on low cohesive classes.

However, software refactoring can be very challenging, especially in large software systems. For this reason, a lot of effort has been devoted to the definition of approaches to support refactoring activities. These approaches can be classified into two different categories: (i) approaches that identify source code components which may require refactoring and (ii) approaches that (semi)automatically perform refactoring operations[2]. These two categories of approaches are complementary. In fact, the former approaches can identify a source code component which requires refactoring (*e.g.*, a low cohesive class), but are not able to suggest refactoring solutions to solve the problem (*e.g.*, how to split the low cohesive class in more meaningful, high cohesive classes). On the other hand, the latter category of approaches can be applied once a source code component which requires refactoring has been identified. This research is related with the latter category of approaches. Among these, some authors have defined approaches able to support different refactoring operations. As instance, O'Keeffe *et al.* [5] formulated the task of refactoring as a search problem in the space of alternative designs resulting from different types of refactoring operations. The search for the optimal design is guided by a quality evaluation function based on structural OO design metrics that reflects refactoring goals.

Other authors have focused their attention on the definition of approaches to support a specific refactoring operation such as extract method refactoring [6], move method refactoring [7], and extract class refactoring [8].

For example, Fokaefs *et al.* [8] use a hierarchical clustering to support extract class refactoring (ECR), a technique to split classes with many responsibilities and low cohesion (known as Blobs) into different classes having higher cohesion. Their approach analyses the structural relationships existing between attributes and methods of a Blob. A hierarchical clustering algorithm is then used to cluster together attributes and methods having strong structural dependencies, extracting them from the Blob class.

Several other refactoring operations have been automated in literature and almost all of them exploit quality metrics based on structural information extracted from source code to suggest refactoring solutions. Recently, semantic metrics have also been proposed [9]. These metrics use semantic information retrieved from comments and identifiers and seem to complement structural metrics [9]. Only in some work on software re-modularization (see *e.g.*, [10]) semantic information has been taken into account.

The orthogonality between structural and semantic metrics motivated us to propose an ECR approach that employs both structural and semantic information to recommend refactoring operations [4]. The proposed approach is based on graph theory and is able to split a class with low cohesion in two classes having a higher cohesion, using a MaxFlow-MinCut algorithm. The empirical evaluation of this approach showed that the combination of structural and semantic measures strongly improve the meaningfulness of the identified ECR operations with respect to the use of only structural or semantic measures [4]. However, this approach suffers of a strong limitation: it always split the original class in two new classes while often the responsibilities

[1] A complete refactoring catalog can be found at http://refactoring.com/.
[2] A survey of the two categories of approaches can be found in [4].

implemented in a Blob class need to be distributed among more than two classes. Such a problem can be solved using partitioning or hierarchical clustering algorithms (see *e.g.*, [8]). Unfortunately, such algorithms suffer of important limitations as well. The former requires as input the number of clusters, *i.e.*, the number of classes to be extracted, while the latter requires a threshold to cut the dendogram. Until now, no heuristics have been derived to suggest good default values for these parameters.

Even if our MaxFlow-MinCut approach suffers of the discussed limitation, its empirical evaluation highlighted the importance of combining structural and semantic information during ECR [4]. This result motivates this research. We conjecture that better refactoring solutions might be identified exploiting semantic information embedded in the source code, besides structural information. To verify our conjecture we (i) further investigated about the usefulness of combining structural and semantic information during ECR proposing a new approach able to split the Blob class in more than two classes, overcoming the limitation of the MaxFlow-MinCut approach and (ii) have used a combination of structural and semantic information to support another refactoring operation, called Extract Package Refactoring (EPR). The goal of EPR is to remove from OO software systems promiscuous-low cohesive packages, decomposing them into smaller and meaningful packages having higher cohesion. Thus, EPR can be used to improve the subsystem decomposition of OO software systems.

II. Supporting Extract Class Refactoring

As the MaxFlow-MinCut approach, also our new ECR approach is based on graph theory. In particular, it represents the Blob to be refactored as a weighted graph, where each node represents a method of the class and the weight of an edge that connects two nodes (methods) represents the likelihood that two methods should be in the same class (*i.e.*, the likelihood that they implement strictly related responsibilities). This likelihood is computed as a hybrid coupling measure between methods obtained by combining three structural and semantic measures, *i.e.*, Structural Similarity between Methods (SSM) [11], Call-based Dependence between Methods (CDM) [4], and Conceptual Similarity between Methods (CSM) [9].

SSM is a structural measure that analyses the instance variables used in methods: the higher the number of instance variables two methods share, the higher the likelihood that two methods should be in the same class. CDM is another structural measure that takes into account the calls performed by the methods. In particular, if two methods exhibit high calls interaction, they should be placed in the same class to reduce coupling between classes. Finally, CSM is a measure based on the semantic information captured in the code by comments and identifiers. The higher the overlap of terms between comments and identifiers of two methods, the higher the likelihood that they implement similar responsibilities (and thus should be placed in the same class).

Since all the exploited measures have values in $[0, 1]$, we compute the overall similarity between two methods m_i and m_j (likelihood that they should be in the same class) as:

$$sim(m_i, m_j) = w_{SSM} \cdot SSM(m_i, m_j) + w_{CDM} \cdot CDM(m_i, m_j) + w_{CSM} \cdot CSM(m_i, m_j)$$

where $w_{SSM} + w_{CDM} + w_{CSM} = 1$ and their values express the confidence (*i.e.*, weight) in each measure.

Once the graph is built, the ECR process is performed in two steps. After filtering out spurious relationships between methods, the approach defines chains of strongly related methods exploiting the transitive closure of the filtered graph. The extracted chains are then refined by merging trivial chains (*i.e.*, chains with very few methods) with non-trivial chains. Using the extracted chains of methods it is possible to create new classes (one for each chain) having higher cohesion than the original class. The attributes of the original class are then distributed among the extracted classes according to how they are used by the methods in the new classes, *i.e.*, each attribute is assigned to the new class having the higher number of methods using it[3].

It is worth noting that this approach (i) is able to split a Blob class in more than two classes, overcoming the limitation of the MaxFlow-MinCut approach, and (ii) on the contrary of the approaches based on clustering algorithms (see *e.g.*, [8]) automatically identifies the appropriate number of classes that should be extracted from a Blob class. More details about this approach can be found in [12], [13].

A. Assessment of the Approach

The approach has been assessed on five open source systems, namely ArgoUML, Eclipse, GanttProject, JHotDraw, and Xerces to (i) analyse the impact of the configuration parameters (*i.e.*, different weights for the similarity measures, and different values for threshold used to remove spurious relationships) on its performances and (ii) verify if the combination of structural and semantic measures improve the meaningfulness of the identified ECR operations with respect to the use of only structural or semantic measures.

To have a high number of classes to assess the proposed approach, we artificially created Blob classes with more responsibilities and low cohesion from classes of the original systems. Specifically, for each system, we randomly created groups of two or three high cohesive classes and merged them to create artificial Blobs, *i.e.*, we created a new class containing all the methods (except the constructors) and attributes of the classes to be merged. For each system we created 50 artificial Blobs composed by merging two classes and 50 by merging three classes.

[3]If a private field needs to be shared by two or more of the extracted classes, the implementation of the needed getter and/or setter methods is left to the developer.

Table I
FM_{avg} ON ECLIPSE USING (I) ONLY STRUCTURAL MEASURES, (II) ONLY SEMANTIC MEASURES, AND (III) A COMBINATION OF BOTH

Information exploited	2 Merged Classes	3 Merged Classes
Structural	0.73	0.61
Semantic	0.82	0.63
Combined	**0.89**	**0.73**

The proposed approach was then applied to split the artificial Blobs and to reconstruct the original classes. In essence, given the observed quality of all the merged classes (very high cohesion), we considered them as a *golden standard*. Hence, to evaluate the results, the refactored classes were compared with the original classes to count the number of methods correctly and incorrectly moved in the split classes. We used the F-measure (FM) [14] to quantify the reconstruction accuracy.

To analyse the impact of the configuration parameters on the performances of the proposed approach we refactored the same set of artificial Blobs using different parameter configurations. In particular, we varied each measure weight starting at 0 and increasing it to 1 in steps of 0.1. In this way it was also possible to verify if the combination of structural and semantic measures (obtained when $w_{SSM} + w_{CDM} \neq 0$ and $w_{CSM} \neq 0$) allows to obtain better performances than those achieved exploiting only structural ($w_{SSM}+w_{CDM} = 1$ and $w_{CSM} = 0$) or semantic ($w_{SSM} + w_{CDM} = 0$ and $w_{CSM} = 1$) measures. We also experimented different values for the threshold used to remove spurious relationships among the methods of the class to be refactored.

The performed assessment allowed us to identify heuristics to set all the parameters of the approach in order to maximize its performances. Moreover, the obtained results highlighted the key role played by the semantic measure in the identification of the refactoring solutions. Table I reports the average FM achieved by our approach on Eclipse when refactoring artificial Blobs composed of two and three merged classes using (i) only structural measures, (ii) only the semantic measure, and (iii) the combination of structural and semantic measures. As we can see, the combination of structural and semantic measures strongly improves the performances of our approach with respect to the use of only structural or semantic measures (confirming the findings of [4]). The results obtained on the other object systems are consistent with those obtained on Eclipse.

See [13] for details about the performed assessment.

B. Evaluation

The proposed approach, once identified its best configuration of parameters, has been used to split 17 real Blobs of two open source systems, namely Xerces and GanttProject. Then, we measured the improvement provided in terms of quality metrics. Our approach provided an average increment of class cohesion for the refactored classes of about 90% against the 75% obtained by MaxFlow-MinCut approach. Moreover, we performed an user study to analyse if the refactoring operations proposed by our approach are meaningful from a developer's point of view. We involved a total of 50 master students that evaluated three different refactoring operations for each of the Blobs used in the experimentation: (i) the refactoring suggested by the transitive closure approach, (ii) the refactoring suggested by the MaxFlow-MinCut approach, and (iii) a random refactoring. The latter option was considered only to verify whether participants seriously considered the assignment. For each of the proposed refactoring the students had to express their level of agreement to the claim "*The proposed refactoring results in a better division of responsibilities*" proposing a score using a Likert scale: 1: Strongly disagree; 2: Disagree; 3: Neutral; 4: Agree; 5: Fully agree.

The transitive closure approach was considered the one providing the most meaningful refactorings with an average score of 4.3 against 3.3 of the MaxFlow-MinCut approach and 1.4 of the random splitting. For details see [13].

III. SUPPORTING EXTRACT PACKAGE REFACTORING

Changing the level of granularity form class to package it is possible to support EPR. Our EPR approach takes as input a package identified by the developer as a candidate for remodularization. Then, a measure reflecting a relationship between the classes of the package is computed. The measured values between classes are stored in a $n \times n$ matrix, called *class-by-class* matrix, where n is the number of classes in the package under analysis. A generic entry $m_{i,j}$ of the class-by-class matrix represents the likelihood that class c_i and class c_j should be in the same package. This likelihood is estimated by combining two structural and semantic measures, *i.e.*, information-flow-based coupling (ICP) [15] and Conceptual Coupling Between Classes (CCBC) [16]. The ICP measures the calls interaction among the classes of the system: the higher the ICP between two classes c_i and c_j, the higher their coupling, the higher the likelihood that c_i and c_j should be placed in the same package in order to reduce coupling among the packages of the system. Concerning the CCBC, it measures the semantic coupling among two classes of the system: the higher the overlap of terms between comments and identifiers of two classes c_i and c_j, the higher the likelihood that these two classes implement similar responsibilities (and thus should be placed in the same package). Once the *class-by-class* matrix is computed, spurious relationships among the classes are removed using a threshold and the transitive closure of the *class-by-class* matrix is exploited to identify the packages that should be extracted from the promiscuous package. Further details about the approach can be found in [17].

A. Evaluation

Also in this case we firstly assessed the performances of the proposed approach in an artificial scenario to identify the

best configuration of parameters for our approach (*i.e.*, the weights to assign to the exploited metrics, and the threshold used to remove spurious relationships). In particular, we selected high cohesive packages from three OO systems and merged them in order to create promiscuous packages in the systems. Then, the proposed approach was applied to reconstruct the original packages. The reconstruction accuracy was once again measured using the F-measure.

The results of the assessment highlighted the importance of the semantic information also to support EPR. In particular, exploiting only structural information, our approach was able to achieve an average reconstruction accuracy of about 35%, while using the combination of structural and semantic information the accuracy reached was of 78% on average.

We also analysed the proposed re-modularization operations from a functional point of view. In particular, an user study was conducted on the same three OO systems. For two of them we were able to involve the original developers in the judgment of the re-modularizations proposed by our approach on the object systems. We asked subjects to evaluate a total of 21 proposed re-modularizations. Among those, the developers marked as non meaningful only 2 re-modularizations, highlighting the goodness of the EPR operations suggested by our approach. For all the details about the experimentation see [17].

IV. Conclusion and Future Work

This research investigates the usefulness of combining structural and semantic information to support refactoring activities. Until now we have exploited structural and semantic information to support two refactoring operations, namely ECR and EPR. The two approaches have been deeply evaluated showing the benefits provided by the semantic information in the identification of refactoring solutions.

Future work will be devoted to the definition of approaches able to support other refactoring operations, like Move Method [18] and Move Class [19]. We are also experimenting other algorithms based on Game Theory [20] to support refactoring activities. Finally, we are implementing our approaches as an Eclipse plug-in.

Acknowledgment

The author would like to thank his advisor Prof. Andrea De Lucia and his co-advisor Prof. Rocco Oliveto for their support and encouragement. The author would also like to thank Prof. Andrian Marcus for his precious contribution to this research.

References

[1] M. Fowler, *Refactoring: improving the design of existing code*. Addison-Wesley, 1999.

[2] T. Mens and T. Tourwe, "A survey of software refactoring," *IEEE TSE*, vol. 30, no. 2, pp. 126–139, 2004.

[3] V. R. Basili, L. Briand, and W. L. Melo, "A validation of object-oriented design metrics as quality indicators," *IEEE TSE*, vol. 22, no. 10, pp. 751–761, 1995.

[4] G. Bavota, A. De Lucia, and R. Oliveto, "Identifying extract class refactoring opportunities using structural and semantic cohesion measures," *Journal of Systems and Software*, vol. 84, pp. 397–414, March 2011.

[5] M. O'Keeffe and M. O'Cinneide, "Search-based software maintenance," *CSMR*, 2006, pp. 249–260.

[6] K. Maruyama and K. Shima, "Automatic method refactoring using weighted dependence graphs," *ICSE*, 1999, pp. 236–245.

[7] O. Seng, J. Stammel, and D. Burkhart, "Search-based determination of refactorings for improving the class structure of object-oriented systems," *GECCO*, 2006, pp. 1909–1916.

[8] M. Fokaefs, N. Tsantalis, A. Chatzigeorgiou, and J. Sander, "Decomposing object-oriented class modules using an agglomerative clustering technique," *ICSM*, 2009, pp. 93–101.

[9] A. Marcus, D. Poshyvanyk, and R. Ferenc, "Using the conceptual cohesion of classes for fault prediction in object-oriented systems," *IEEE TSE*, vol. 34, pp. 287–300, 2008.

[10] A. Kuhn, S. Ducasse, and T. Gîrba, "Semantic clustering: Identifying topics in source code," *Information and Software Technology*, vol. 49, no. 3, pp. 230–243, 2007.

[11] G. Gui and P. D. Scott, "Coupling and cohesion measures for evaluation of component reusability," *MSR*, 2006, pp. 18–21.

[12] G. Bavota, A. D. Lucia, A. Marcus, and R. Oliveto, "A two-step technique for extract class refactoring," *ASE*, 2010, pp. 151–154.

[13] ——, "Automating extract class refactoring. Technical Report. www.sesa.dmi.unisa.it/tr/trECR11.pdf."

[14] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. Addison-Wesley, 1999.

[15] Y. Lee, B. Liang, S. Wu, and F. Wang, "Measuring the coupling and cohesion of an object-oriented program based on information flow," *International Conference on Software Quality*, 1995, pp. 81–90.

[16] D. Poshyvanyk, A. Marcus, R. Ferenc, and T. Gyimóthy, "Using information retrieval based coupling measures for impact analysis," *Empirical Software Engineering*, vol. 14, no. 1, pp. 5–32, 2009.

[17] G. Bavota, A. D. Lucia, A. Marcus, and R. Oliveto, "Software re-modularization based on structural and semantic metrics," *WCRE*, 2010, pp. 195–204.

[18] R. Oliveto, M. Gethers, G. Bavota, D. Poshyvanyk, and A. D. Lucia, "Identifying method friendships to remove the feature envy bad smell (nier track)," *ICSE*, 2011, pp. 820–823.

[19] G. Bavota, M. Gethers, R. Oliveto, D. Poshyvanyk, and A. D. Lucia, "Improving software modularization via automated analysis of latent topics and dependencies. Technical Report. www.sesa.dmi.unisa.it/tr/trMC11.pdf."

[20] G. Bavota, R. Oliveto, A. D. Lucia, G. Antoniol, and Y.-G. Guéhéneuc, "Playing with refactoring: Identifying extract class opportunities through game theory," *ICSM*, 2010.