ARIES: An Eclipse plug-in to Support Extract Class Refactoring

Gabriele Bavota
University of Sannio
gbavota@unisannio.it

Andrea De Lucia
University of Salerno
adelucia@unisa.it

Andrian Marcus
Wayne State University
amarcus@wayne.edu

Rocco Oliveto
University of Molise
rocco.oliveto@unimol.it

Fabio Palomba
University of Salerno
fabio.palomba.89@gmail.com

Michele Tufano
University of Salerno
tufanomichele89@gmail.com

ABSTRACT

During Object-Oriented development, developers try to define classes having (i) strongly related responsibilities, i.e., high cohesion, and (ii) limited number of dependencies with other classes, i.e., low coupling [1]. Unfortunately, due to strict deadlines, programmers do not always have sufficient time to make sure that the resulting source code conforms to such a development laws [8].

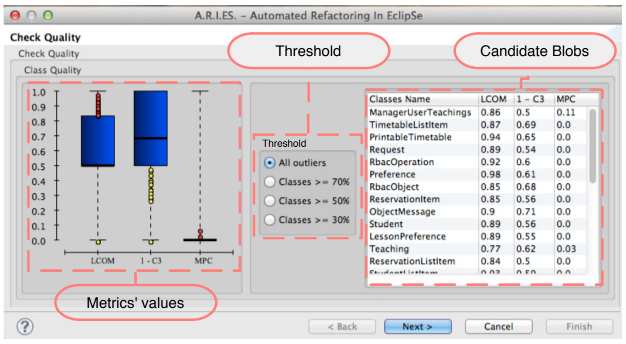
In particular, during software evolution the internal structure of the system undergoes continuous modifications that makes the source code more complex and drifts away from its original design. Classes grow rapidly because programmers often add a responsibility to a class thinking that it is not required to include it in a separate class. However, when the added responsibility grows and breeds, the class becomes too complex and its quality deteriorates [8]. A class having more than one responsibility has generally low cohesion and high coupling. Several empirical studies provided evidence that high levels of coupling and lack of cohesion are generally associated with lower productivity, greater rework, and more significant design efforts for developers [6], [10], [11], [12], [13]. In addition, classes with lower cohesion and/or higher coupling have been shown to correlate with higher defect rates [9], [14], [15].

Classes with unrelated methods often need to be restructured by distributing some of their responsibilities to new classes, thus reducing their complexity and improving their cohesion. The research domain that addresses this problem is referred to as refactoring [8]. In particular, *Extract Class* Refactoring allows to split classes with many responsibilities into different classes. Moreover, it is a widely used technique to address the Blob antipattern [8], namely a large and complex class, with generally low cohesion, that centralize the behavior of a portion of a system and only use other classes as data holders. It is worth noting that performing Extract Class Refactoring operations manually might be very difficult, due to the high complexity of some Blobs. For this reason, several approaches and tools have been proposed to support this kind of refactoring. Bavota et. al [2] proposed an approach based on graph theory that is able to split a class with low cohesion into two classes having a higher cohesion, using a MaxFlow-MinCut algorithm. An important limitation of this approach is that often classes need to be split in more than two classes. Such a problem can be mitigated using partitioning or hierarchical clustering algorithms. However, 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 the definition of a threshold to cut the dendogram. Unfortunately, no heuristics have been derived to suggest good default values for all these parameters. Indeed, in the tool JDeodorant [7], which uses a hierarchical clustering algorithm to support Extract Class Refactoring, the authors tried to mitigate such an issue by proposing different refactoring opportunities that can be obtained using various thresholds to cut the dendogram. However, such an approach requires an additional effort by the software engineer who has to analyze different solutions in order to identify the one that provides the most adequate division of responsibilities.

We tried to mitigated such deficiencies by defining an approach able to suggest a suitable decomposition of the original class by also identifying the appropriate number of classes to extract [3, 4]. Given a class to be refactored, the approach calculates a measure of cohesion between all the possible pairs of methods in the class. Such a measure captures relationships between methods that impact class cohesion (e.g., attribute references, method calls, and semantic content). Then, a weighted graph is built where each node represents a method and the weight of an edge that connects two nodes is given by the cohesion of the two methods. The higher the cohesion between two methods the higher the likelihood that the methods should be in the same class. Thus, a cohesion threshold is applied to cut all the edges having cohesion lower than the threshold in order to reduce 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 few methods) with non trivial chains. Exploiting the extracted chains of methods it is possible to create new classes - one for each chain - having higher cohesion than the original class.

In this paper, we present the implementation of the proposed Extract Class Refactoring method in ARIES (Automated Refactoring In EclipSe) [5], a plug-in to support refactoring operations in Eclipse. ARIES provides support for Extract Class Refactoring through a three steps wizard.

In the first step, shown in figure 1, the tool supports the software engineer in the identification of candidate Blobs through the computing of three quality metrics, namely LCOM5 [6], C3 [9] and MPC [16]. Thus, ARIES does not compute an overall quality of the classes, but it considers



Threshold: [✓ All outliers] Classes >= 70% Classes >= 50% Classes >= 30%

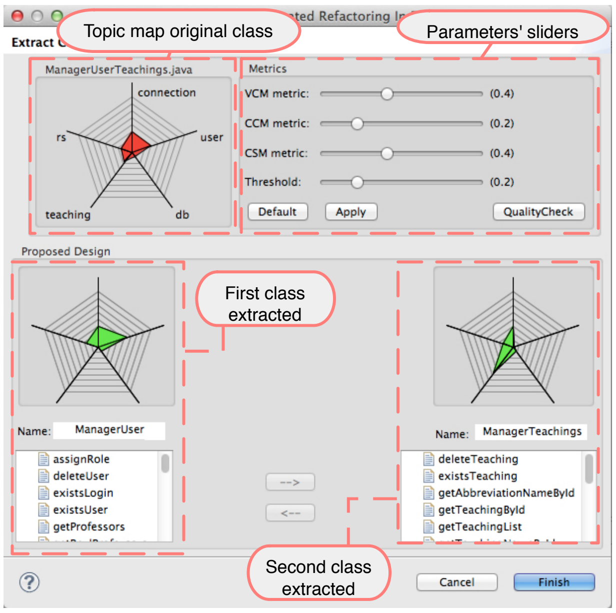
Classes Name	LCOM	1 – C3	MPC
ManagerUserTeachings	0.86	0.5	0.11
TimetableListItem	0.87	0.69	0.0
PrintableTimetable	0.94	0.65	0.0
Request	0.89	0.54	0.0
RbacOperation	0.92	0.6	0.0
Preference	0.98	0.61	0.0
RbacObject	0.85	0.68	0.0
ReservationItem	0.85	0.56	0.0
ObjectMessage	0.9	0.71	0.0
Student	0.89	0.56	0.0
LessonPreference	0.89	0.55	0.0
Teaching	0.77	0.62	0.03
ReservationListItem	0.84	0.5	0.0

Figure 1: ARIES: Identification of candidate Blobs.

only cohesion and coupling as the main indicators of class quality in this context. Hence, Blobs are usually outliers or classes having a quality much lower than the average quality of the system under analysis [9]. The identification of Blobs in ARIES is based on such a conjecture. In the second step of the wizard, the software engineer has the possibility to further analyze a candidate Blob and get insights on the different responsibilities implemented by analyzing its topic map, represented as the five most frequent terms in a class (the terms present in the highest number of methods). For this reason, the topic map is represented by a pentagon where each vertex represents one of the main topics. Once a class that needs to be refactored is identified, the software engineer activates the last step of the wizard (shown in figure 2) to obtain a possible restructuring of the class under analysis. ARIES reports for each class that should be extracted from the Blob the following information: (i) its topic map; (ii) the set of methods composing it; and (ii) a text field where the developer can assign a name to the class. The tool also allows the developer to customize the proposed refactoring moving the methods between the extracted classes.

In addition, ARIES offers the software engineer on-demand analysis of the quality improvement obtained by refactoring the Blob, by comparing various measures of the new classes with the measures of the Blob. When the developer ends the analysis, the extraction process begins. ARIES will generate the new classes making sure that the changes made by the refactoring do not introduce any syntactic error. A video of the tool is available on Youtube[1].

[1]http://www.youtube.com/watch?v=csfNhgJlhH8



VCM metric: (0.4)
CCM metric: (0.2)
CSM metric: (0.4)
Threshold: (0.2)

Name: ManagerUser
assignRole
deleteUser
existsLogin
existsUser
getProfessors

Name: ManagerTeachings
deleteTeaching
existsTeaching
getAbbreviationNameById
getTeachingById
getTeachingList

Figure 2: ARIES: Extract Class refactoring.

REFERENCES

[1] W. Stevens, G. Myers, and L. Constantine. Structured design. IBM Systems Journal, vol. 13, no. 2, pp. 115139, 1974.

[2] G. Bavota, A. De Lucia, and R. Oliveto. Identifying extract class refactoring opportunities using structural and semantic cohesion measures. *JSS*, 84:397–414, 2011.

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

[4] G. Bavota, A. D. Lucia, A. Marcus, and R. Oliveto. Automating Extract Class Refactoring: an Improved Method and its Evaluation. Empirical Software Engineering (EMSE) (2013) To appear.

[5] G. Bavota, A. D. Lucia, A. Marcus, R. Oliveto and F. Palomba. Supporting Extract Class Refactoring in Eclipse: The ARIES Project. *ICSE*, 1419–1422, 2012.

[6] S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. *IEEE TSE*, 20(6):476–493, 1994.

[7] M. Fokaefs, N. Tsantalis, E. Stroulia, A. Chatzigeorgiou. JDeodorant: identification and application of extract class refactorings. *ICSE*, 1037–1039, 2011.

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

[9] A. Marcus, D. Poshyvanyk, and R. Ferenc. Using the conceptual cohesion of classes for fault prediction in object-oriented systems. *IEEE TSE*, 34(2):287–300, 2008.

[10] 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

[11] A. B. Binkley and S. R. Schach. Validation of the coupling dependency metric as a predictor of run-time failures and maintenance measures. *ICSE*, 452–455, 1998

[12] L. C. Briand, J. Wüst, and H. Lounis. Using coupling measurement for impact analysis in object-oriented systems. *ICSM*, 475–482, 1999

[13] L. C. Briand, J. Wüst, S. V. Ikonomovski, and H. Lounis. Investigating quality factors in object-oriented designs: an industrial case study. *ICSE*, 345–354, 1999

[14] T. Gyimóthy, R. Ferenc, and I. Siket. Empirical validation of object-oriented metrics on open source software for fault prediction. *IEEE TSE*, vol. 31, no. 10, pp. 897910, 2005

[15] Y. Liu, D. Poshyvanyk, R. Ferenc, T. Gyimóthy, and N. Chrisochoides. Modelling class cohesion as mixtures of latent topics. *ICSM*, 233–242

[16] W. Li and S. Henry. Maintenance metrics for object oriented paradigm. *Software Metrics Symposium*, 52–60