Supporting Extract Class Refactoring in Eclipse: The ARIES Project

Gabriele Bavota[1], Andrea De Lucia[1], Andrian Marcus[2], Rocco Oliveto[3], Fabio Palomba[3]
[1]*School of Science, University of Salerno, 84084 Fisciano (SA), Italy*
[2]*Computer Science Department, Wayne State University, Detroit, MI 48202, USA*
[3]*STAT Department, University of Molise, 86090 Pesche (IS), Italy*
gbavota@unisa.it, adelucia@unisa.it, amarcus@wayne.edu, rocco.oliveto@unimol.it, fabio.palomba.89@gmail.com

***Abstract*—During software evolution changes are inevitable. These changes may lead to design erosion and the introduction of inadequate design solutions, such as *design antipatterns*. Several empirical studies provide evidence that the presence of antipatterns is generally associated with lower productivity, greater rework, and more significant design efforts for developers. In order to improve the quality and remove antipatterns, refactoring operations are needed. In this demo, we present the Extract class features of ARIES (Automated Refactoring In EclipSe), an Eclipse plug-in that supports the software engineer in removing the "Blob" antipattern.**

***Keywords*-Refactoring; Design; Quality**

I. Introduction

During software evolution change is the rule rather than the exception [6]. Unfortunately, such changes are usually performed by developers that due to strict deadlines do not have enough time to make sure that every change conforms to OOP guidelines, such as, minimizing coupling and maximizing cohesion of classes. Such careless design solutions often lead to *design antipatterns*, which negatively impact the quality of a software system, making its maintenance difficult and expensive. An example of antipattern is represented by the *Blob* [6]. Blobs are large and complex classes, with generally low cohesion, that centralize the behavior of a portion of a system and only use other classes as data holders. Such characteristics make the maintenance of Blobs difficult (high effort required to comprehend the class) and dangerous (low cohesive classes are more error-prone than other classes [10]).

The *Extract Class* Refactoring (ECR) is a widely used technique to address the Blob antipattern [6]. ECR aims at restructuring Blobs by distributing some of their responsibilities to new classes, thus reducing their complexity and improving their cohesion. It is worth noting that performing ECR 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 the ECR. Bavota et. al [1] 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 [5], which uses a hierarchical clustering algorithm to support ECR, 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 mitigate 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 [2], [3]. Given a class to be refactored, the approach builds a weighted graph where each node represents a method of the class and the weight of an edge that connects two nodes is given by the similarity of the two methods. This is a composite measure that captures different types of relationships between methods, which impact class cohesion and coupling. After filtering out spurious relationships between methods, the approach defines chains of strongly related methods using the transitive closure of the filtered graph. Using 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 demo, we present the implementation of the proposed ECR method in ARIES, a plug-in to support refactoring operations in Eclipse. ARIES provides support for ECR through a three steps wizard. In the first step, the tool supports the software engineer in the identification of candidate Blobs through the analysis of several quality metrics. In the second step, 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. Once a class that needs to be refactored is identified, the software engineer activates the last step of the wizard to obtain a possible restructuring of the class under analysis. As before, topic maps of the extracted classes are used to give insight on how ARIES

splits the responsibilities of the original Blob in the new 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. A video of the tool is available on Youtube[1].

II. The Extract Class Feature of ARIES

This section describes the Extract Class feature of ARIES The current implementation of the tool supports Java (as it is integrated in the Eclipse Java Development Kit) although the proposed approach can be extended to other OO programming languages.

A. The Extract Class Refactoring Algorithm

The ECR technique implemented by ARIES [2], [3] is able to extract two or more classes from a given class with several responsibilities, i.e., a *Blob*. In particular, the input class is first parsed to build a *method-by-method* matrix, a $n \times n$ matrix where n is the number of methods in the class to be refactored. A generic entry $c_{i,j}$ of the *method-by-method* matrix represents the likelihood that method m_i and method m_j should be in the same class. This likelihood is computed as a hybrid coupling measure between methods (degree to which they are related) obtained through a weighted average of three structural and semantic measures, i.e., the Structural Similarity between Methods (SSM) [7], the Call-based Dependence between Methods (CDM) [1], and the Conceptual Similarity between Methods (CSM) [10].

Once the *method-by-method* matrix has been constructed, its transitive closure is computed in order to extract chains of strongly related methods (each chain represents the set of responsibilities, i.e., methods, that should be grouped in a new class). However, in the *method-by-method matrix* there might be very few zero values, due to spurious (but light) structural and/or semantic relationships between methods [1]. Thus, a transitive closure might include almost all the methods in a single chain. To avoid such a problem and to identify the strongest relationships between methods, the *method-by-method matrix* is filtered based on a threshold $minCoupling$, i.e., all similarity values less than the threshold $minCoupling$ are converted to zero. One possible side effect is that some of the extracted chains might be very short. To avoid the extraction of trivial classes with a very low number of methods, a length threshold $minLength$ is used and each chain shorter than $minLength$ is merged with the most similar non trivial chain (see [2] for more details). By default $minLength = 3$ since it is unlikely that classes with a well-defined set of responsibilities would have less than three methods.

As final step of the class extraction process, the attributes of the original class are distributed among the extracted

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

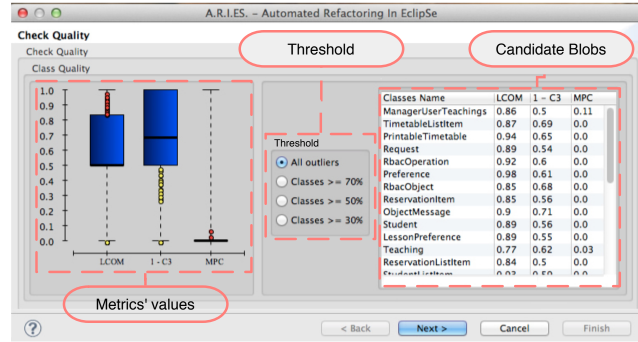

Figure 1. ARIES: Identification of candidate Blobs.

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. 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 automatically done by ARIES.

The proposed approach has been empirically evaluated through two studies [3]. The goal of the first study was to assess the parameters of the approach. Based on the results of the first study, the second study involved 50 Master students who analyzed refactoring operations suggested by the proposed approach on the Blobs of two open source systems. The goal of this second study was to analyze whether: (i) the extract classes have higher quality than the original classes, and (ii) the extracted classes make sense from a functional point of view. The results indicated that the classes extracted with the proposed approach have higher cohesion than the original classes and are meaningful from a functional point of view.

B. ARIES at Work

ARIES supports ECR with a three steps wizard. In the first two steps the tool provides support to the software engineer to identify and analyze Blobs in the system under analysis. In the third step the software engineer receives recommendations on how to refactor the candidate Blobs. The following sections present details on the three steps of the ECR process in ARIES.

1) Identifying Candidate Blobs: ARIES supports the software engineer in the detection of Blobs through a quality check of the entire software system (or of a specified subsystem). Note that ARIES does not compute an *overall* quality of the classes, but it considers 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 [8]. The identification of Blobs in ARIES is based on such a conjecture.

The software engineer starts the quality check selecting the *Check Quality* command in the main menubar. ARIES computes three quality metrics for each class of the (sub)system, namely, Lack of Cohesion of Methods

(LCOM5) [4], Conceptual Cohesion of Classes (C3) [10], and Message Passing Coupling (MPC) [9]. The results are shown to the developers as three boxplots (one for each quality metric) highlighting any negative outlier, i.e., classes having cohesion (coupling) markedly lower (higher) than the other classes of the system (see Figure 1). Note that for the C3 metric ARIES shows the values of $1 - C3$. In this way for all the three measures the negative outliers are reported on the top of the boxplot.

All the outliers are reported in the list shown in the right side of Figure 1. Note that the list will include all the classes that are negative outliers for at least one of the three metrics. In case no outliers are identified, ARIES allows to "relax" the process used to identify candidate Blobs. In particular, instead of a statistical identification of the outliers, the software engineer can select a threshold $\lambda \in \{70, 50, 30\}$ that allows to recover as candidate Blobs all the classes having a quality (in terms of the employed quality metrics) lower than $\lambda\%$ of the average quality. In the scenario shown in Figure 1 the software engineer decides to analyze the quality of the system in order to identify Blobs. ARIES shows the boxplots for the values of LCOM, C3 and MPC. The software engineer decides to improve the quality of some classes having a quality (in terms of cohesion and coupling) sensibly worse than the average quality of the system. In the top of the list of outliers there is the class `ManagerUserTeaching` that seems to be a good candidate for refactoring. In order to obtain a detailed view on `ManagerUserTeaching`, the developer selects the class from the list and clicks on the "Next" button activating the second step of the wizard.

Note that the *Identification* step is not mandatory in order to perform ECR. The developer can directly select a class in the Package Explorer and start the class extraction process by clicking the ECR button in the main toolbar.

2) Analyzing Candidate Blobs: The second step of the ARIES wizard aims at helping the software engineer in better analyzing the classes that are candidates for refactoring. The tool shows the preview of the class under analysis as well as its topic map (see Figure 2). The topic map of a generic class C is built by analyzing the term frequency in the methods it contains. The five most frequent terms (the terms present in the highest number of methods) are used to construct the topic map of C that, for this reason, is represented by a pentagon where each vertex represents one of the main topics. Each vertex is connected to the center of the pentagon by an axis representing the percentage of methods in C that implements the corresponding topic. The graphical representation of the main topics of C is then obtained by tracing lines between the percentage points on each of the five axes indicating the percentage of methods belonging to C that implement the corresponding topic. Note that a stop-word list is used to automatically prune out common English words and Java keywords. This stop-

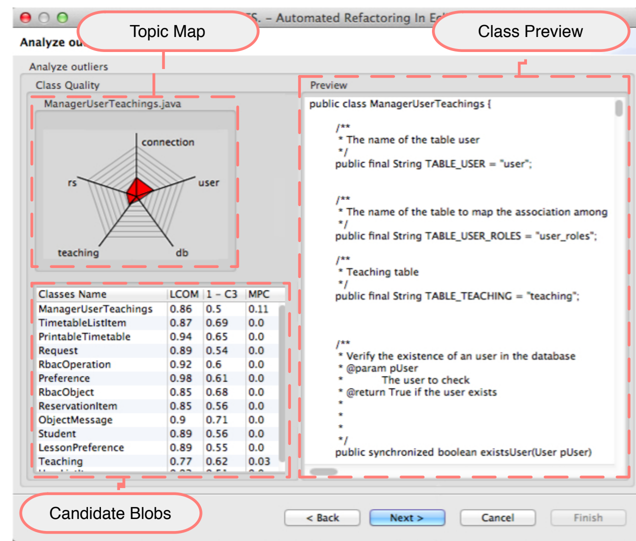

Figure 2. ARIES: Analysis of candidate Blobs.

word list can be customized using the ARIES preference panel. The topic map provided for the Blob is meant to help the developer in understanding which are the different responsibilities implemented in the class. Clearly not all topic maps will be equally helpful, as they depend on identifiers and comments in the code.

In the scenario shown in Figure 2 the software engineer is analyzing the class `ManagerUserTeaching`. From the analysis of the topic map it is easy to identify three different responsibilities of the class, i.e., database connection (indicated by the terms `connection`, `db`, and `rs`), user management (indicated by the term `user`), and teaching management (indicated by the term `teaching`). While the first responsibility is a common for each control class (each control class in the analyzed system accesses the database to manage particular information), the other two responsibilities are quite different suggesting that the quality of the class (in terms of cohesion) could be improved splitting it in different classes.

3) Refactoring the Blobs: Figure 3 shows the final step of the ECR feature of ARIES. The upper part of Figure 3 contains the topic map of the class to be refactored. The right part contains all the sliders to configure parameters of the ECR approach, i.e., the weights for the similarity measures and the threshold $minCoupling$ (see Section II-A). Although initial default values achieved through an empirical assessment [3] are provided to the developer, she can modify any parameter, changing on-the-fly the resulting refactoring recommendation, shown in the bottom part of Figure 3. 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.

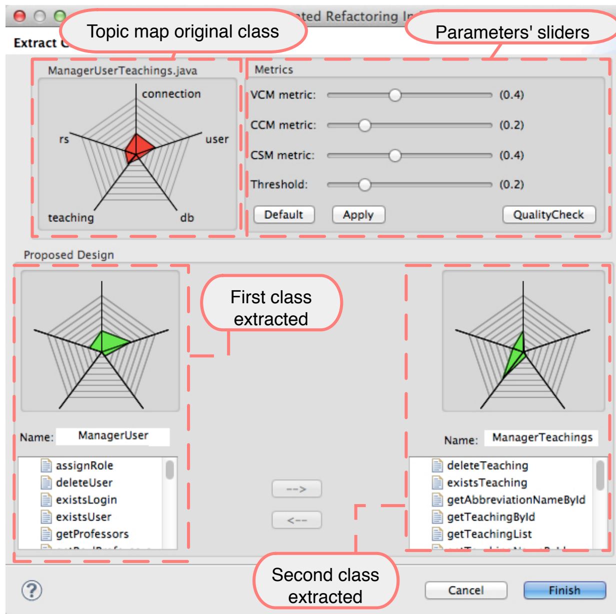

Figure 3. ARIES: Extract Class refactoring.

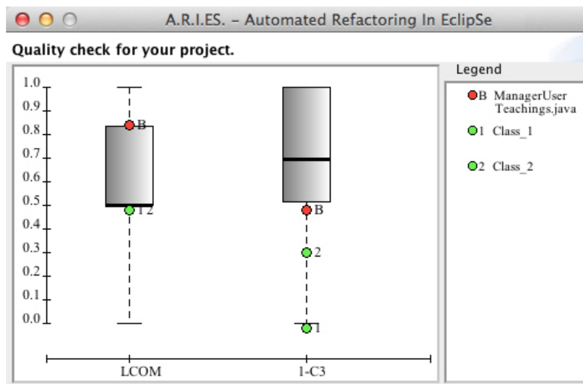

Figure 4. ARIES: Quality Check of the refactoring operation

In the scenario of Figure 3, ARIES splits the class `ManagerUserTeaching` into two classes. The topic maps of the extracted classes help to understand the rationale behind the refactoring recommendation. The first class is in charge of managing the users, while the second class is responsible of teaching management. From the analysis of the topic maps it is also possible to see that database connection is a responsibility of both classes. This means that both classes access the database to manage users and teachings, respectively.

Besides a conceptual analysis (based on the topic maps) of the refactoring proposed by ARIES, the developer can quantify the quality improvement obtained applying the proposed refactoring. Using the functionality "Quality Check", ARIES highlights on the boxplots of the metrics showed in the first step of the wizard, the values of the metrics for the new classes and the original Blob (see Figure 4). In this way the developer can analyze the quality of the new classes as compared to the overall quality of the system.

To terminate the extraction process and automatically generate the new classes, the software engineer can click the "Finish" button (see right lower corner in Figure 3). ARIES will generate the new classes making sure that the changes made by the refactoring do not introduce any syntactic error.

III. Demo Remarks

In this demo we presented ARIES, an ECR tool that supports the software engineer in the identification, analysis, and resolution of the Blob antipattern. As future work, we plan (i) to extend the functionalities of the tool by implementing approaches supporting other refactoring operations (e.g., Move Method [12]) and (ii) to implement a more sophisticated approach to detect Blob classes in a software system (see e.g., [11]).

Acknowledgment

The authors would like to thank Veronica D'Uva for her constructive comments on the preliminary version of ARIES.

Andrian Marcus was supported in part by grants from the US NSF - CCF1017263 and CCF0845706.

References

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

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

[3] G. Bavota, A. D. Lucia, A. Marcus, and R. Oliveto. Automating extract class refactoring: a novel approach and its evaluation. Technical report, http://www.distat.unimol.it/reports/ARIES/ECR.pdf

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

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

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

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

[8] M. Lanza and R. Marinescu. *Object-Oriented Metrics in Practice*. Springer, 2006.

[9] W. Li and S. Henry. Maintenance metrics for object oriented paradigm. *Software Metrics Symposium*, 52–60, 1993.

[10] 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.

[11] N. Moha, Y.-G. Gueheneuc, L. Duchien, and A.-F. L. Meur. Decor: A method for the specification and detection of code and design smells. *IEEE TSE*, 36:20–36, 2010.

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