Extract Package Refactoring in ARIES

Fabio Palomba*, Michele Tufano†, Gabriele Bavota‡, Rocco Oliveto§,
Andrian Marcus¶, Denys Poshyvanyk†, and Andrea De Lucia*
*Department of Management & Information Technology, University of Salerno
†Department of Computer Science, The College of William & Mary, VA, USA
‡Center for Applied Software Engineering, Free University of Bolzano-Bozen
§Department of Bioscience and Territory, University of Molise
¶Department of Computer Science, University of Texas

***Abstract*—Software evolution often leads to the degradation of software design quality. In Object-Oriented (OO) systems, this often results in packages that are hard to understand and maintain, as they group together heterogeneous classes with unrelated responsibilities. In such cases, state-of-the-art re-modularization tools solve the problem by proposing a new organization of the existing classes into packages. However, as indicated by recent empirical studies, such approaches require changing thousands of lines of code to implement the new recommended modularization. In this demo, we present the implementation of an Extract Package refactoring approach in ARIES (Automated Refactoring In EclipSe), a tool supporting refactoring operations in Eclipse. Unlike state-of-the-art approaches, ARIES automatically identifies and removes single low-cohesive packages from software systems, which represent localized design flaws in the package organization, with the aim to incrementally improve the overall quality of the software modularisation.**
URL: http://www.sesa.unisa.it/tools/aries.jsp

I. Introduction

During software maintenance and evolution, changes are inevitable [1]. A software system evolves in order to be adapted to new environments and/or requirements. One of the most important tasks in this phase concerns the management of software complexity. In Object-Oriented (OO) software, packages group together logically and structurally related classes with the aim of localizing changes, among other things. Such groupings should be done with care, otherwise they result in modules that are hard to understand and maintain. Researchers defined coupling and cohesion as properties of decomposition units and generally accepted rules state that modules (i.e., classes and packages in OO software systems) should have high cohesion and low coupling.

During software evolution the structural design of software systems changes [1]. Such changes are driven by the processes (if any) used by the developers, their individual choices, and by external pressure. In consequence, more often than not, the software design quality decreases over time. In such cases a re-modularization of the system is recommended. However, as recently highlighted by Hall *et al.* [2], "Big-Bang" re-modularization (i.e., a complete re-organization of the system's classes into packages) is not a viable solution since even for modest systems, the number of changes required to apply it is in the order of thousands of lines of code [2]. Thus, re-modularization efforts need to be localized and aimed at solving specific design issues.

The Extract Package Refactoring (EPR) can be particularly useful to avoid "Big-Bang" re-modularization and incrementally improve the overall quality of the system modularisation. EPR aims at restructuring *promiscuous package*, i.e., low-cohesive packages grouping together classes implementing heterogeneous and unrelated responsibilities, by distributing some of their responsibilities to new packages, thus reducing their complexity and improving their cohesion. Performing EPR operations manually can be very difficult, due to, for example, the high number of classes contained in a package.

In this demo, we present the implementation of an Extract Package refactoring approach in ARIES (**A**utomated **R**efactoring **I**n **E**clip**S**e), a tool supporting refactoring operations in Eclipse. Specifically, we provided to ARIES functionalities to support both the identification and the removal of promiscuous packages. The tool is composed by a two-step wizard. In the first step, ARIES provides to the developer a list of candidate promiscuous packages to refactor by analyzing the quality metric profile of the packages of the system under analysis. In the second step the tool proposes a possible re-organization of the candidate package. Topic maps are used to allow developers to better understand the responsibilities implemented in the promiscuous package and in the proposed new packages. A video showing ARIES in action is available on the ARIES website[1], where one can also download the tool.

The rest of the paper is organized as follows. Section II present the tool functionalities, while Section III describes in details the approach used by ARIES to identify and remove promiscuous packages. Finally, Section V concludes the paper highlighting future directions.

II. Extract Package Refactoring in ARIES

ARIES is built on top of the Eclipse Java Development Toolkit (JDT). The first version of ARIES was presented in 2012 and it supported only Extract Class refactoring [3] operations. In this demo, we present the implementation in ARIES of an approach to support another refactoring operation, i.e., the EPR. In the next subsections we provide an overview of how the EPR process works in ARIES, while details about the approach behind this process are presented in Section III.

[1]http://www.sesa.unisa.it/tools/aries.jsp

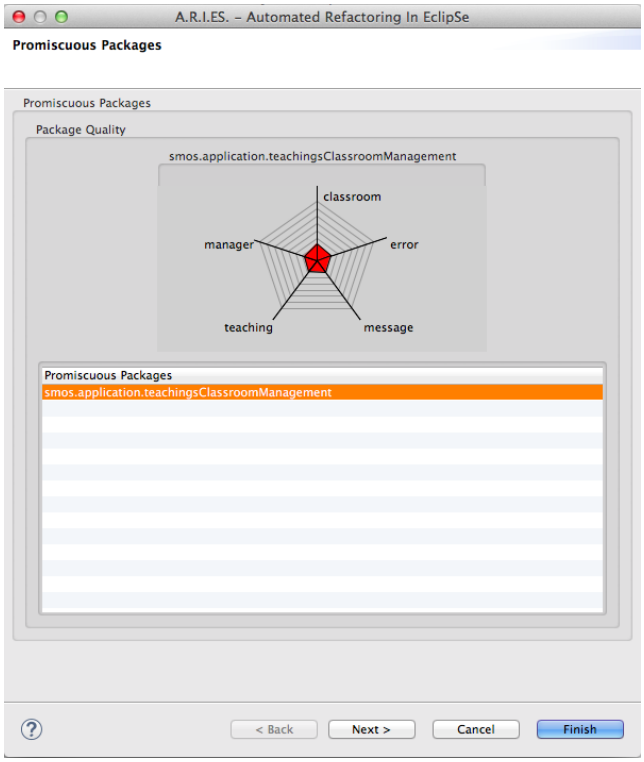

Fig. 1. Identification of Promiscuous Packages in ARIES

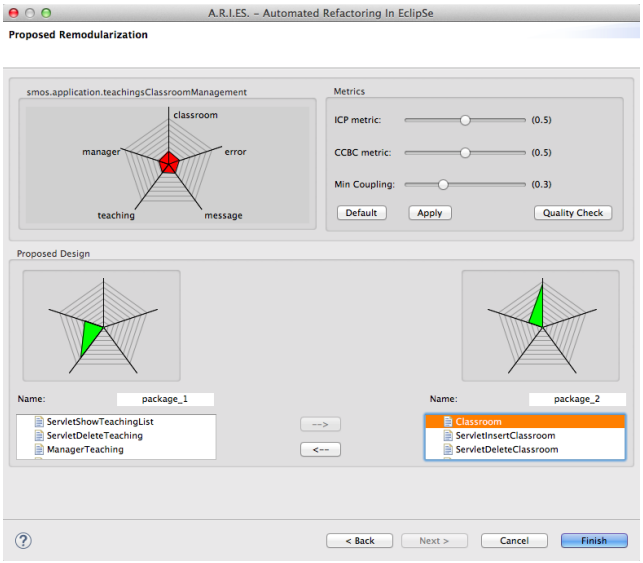

Fig. 2. Refactoring of a Promiscuous Package in ARIES

A. *Identification and Analysis of Promiscuous Packages*

ARIES supports the developer in the identification of promiscuous packages through a quality check of all the packages in the system under analysis (or of a specified package). ARIES considers as indicators of package quality the coupling between each pair of classes in the package, computed by combining the Information flow-based coupling (ICP) [4] and the Conceptual Coupling Between Classes (CCBC) [5] metrics. In particular, a high-quality package should exhibit high (structural and conceptual) coupling between the classes present in it (i.e., the classes implement related responsibilities, thus leading to high package cohesion). Thus, packages containing classes poorly related to each other are identified by ARIES as candidate for EPR.

To start the quality check, the developer selects from the Eclipse main menubar the *Check Quality of Packages* command. Figure 1 shows ARIES at work in a scenario where the developer has requested the analysis of the system he is developing, namely SMOS. SMOS is a software developed for high schools, which offers a set of functionalities aimed at simplifying the communications between the school and the parents of the students.

ARIES shows to the developer a window where, in the lower part, there is a table with all the promiscuous packages instances identified by ARIES, while in the upper part, there is the topic map of the candidate promiscuous package selected from the list. The topic map is a diagram summarizing the concepts (through keywords) implemented in the package. Such a diagram is particularly useful to provide a visual indication to the developer on which are the responsibilities implemented by the package under analysis.

In the scenario depicted in Figure 1, ARIES has identified a single potentially promiscuous package, `teachingClassroomManagement`. The topic map reveals that in this package there are classes that treat different concepts, namely, (i) teaching of the school; and (ii) classrooms. The other topics (keywords) highlighted in the topic map, i.e., `manager`, `message`, and `error`, represent common terms of classes that are involved in the managing the persistence of objects in SMOS.

After the analysis of the topic map, the developer can select the package and ask ARIES to suggest a new re-organization of the classes contained in it, by clicking on the *Next* button in the lower part of the window.

B. *Refactoring Promiscuous Packages*

Figure 2 shows the re-modularization proposed by ARIES. The window is organized in two parts. The upper part contains the topic map (in the left side) of the promiscuous package to be refactored and sliders (in the right side) allowing the developer to set some parameters of the EPR algorithm implemented by ARIES (see Section II). The lower part of the window describes the re-modularization proposed by ARIES. For each package that should be extracted from the promiscuous package, ARIES reports, (i) its topic map; (ii) the set of classes composing it; and (iii) a text field where the developer can assign a name to the package to be created. ARIES also allows the developer to customize the proposed refactoring by moving the classes between the extracted packages. In addition, changing the approach parameters is possible to have a more or less conservative proposed re-modularization, i.e., a remodularization with lower or higher number of extracted packages, respectively.

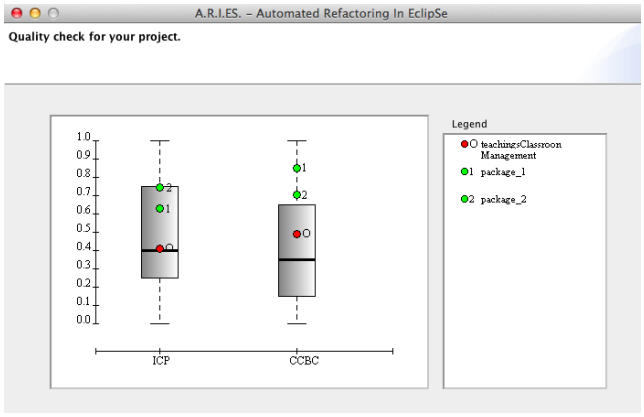

Fig. 3. Functionality *Quality Check* in ARIES

In the scenario depicted in Figure 2, ARIES recommends to split the package `teachingClassroomManagement` into two new packages, i.e., a first one grouping together the classes related to the management of teaching and a second one related to the management of classrooms. This information can be easily derived by the software engineer by analyzing the topic maps of the recommended packages. To further check the quality improvement of the proposed refactoring, the developer can use the *Quality Check* feature implemented in ARIES (see Figure 3). Specifically, ARIES highlights the average values of the ICP and CCBC metrics for the pairs of classes contained in the new packages (green dots in Figure 3) and for the original promiscuous package (red dot in Figure 3) on boxplots reporting the distribution of such metrics for all the packages in the system. In this way the developer can analyze the quality of the new packages as compared to (i) the quality of the original promiscuous package, and (ii) the overall quality of all the packages in the system.

To terminate the extraction process and automatically generate the new packages, the developer can click the *Finish* button (right lower corner in Figure 2). ARIES will generate the new packages making sure that the changes made by the refactoring do not introduce any syntactic error.

III. Tool Architecture

Figure 4 shows the architecture of ARIES. The *Presentation layer* contains the classes implementing the *GUI* of ARIES, exploiting the SWT tools provided by the Eclipse API. All GUIs presented in the previous section are part of this layer.

The *Application logic layer* is the core of ARIES and it contains all the subsystems implementing the identification of design problems and the recommendation of refactoring operations. The Extract Class Refactoring (ECR) subsystem is not detailed in this demo, since we presented it in the first version of ARIES [3]. Our focus is on the EPR subsystem, which is new in ARIES. It includes two main components: the *Promiscuous packages detector* and the *EPR recommender*.

Promiscuous packages detector. This subsystem is in charge of providing the developer with indications on where

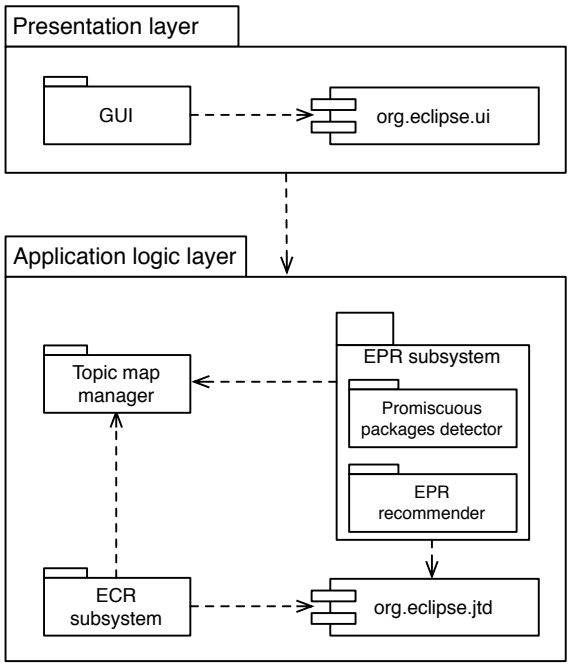

Fig. 4. ARIES Architecture

in her system promiscuous packages lie. As previously mentioned, the detection of promiscuous packages is based on the analysis of two quality metrics, i.e., the ICP [4] and the CCBC [5]. These two metrics capture the coupling between a pair of classes (C_i, C_j) from a structural (ICP) and a conceptual (CCBC) point of view. In particular, the ICP between two classes C_i and C_j is measured as the number of method invocations in the class C_i to methods in the class C_j, weighted by the number of parameters of the invoked methods. CCBC is instead based on the conceptual information (i.e., textual information) captured in the code by comments and identifiers. Two classes are conceptually related if their (domain) semantics are similar, that is, they have high textual similarity. Higher CCBC values indicate higher coupling. Since both metrics are defined between zero and one, ARIES combines them in a composite coupling metric defined as:

$$coupling(C_i, C_j) = \frac{ICP(C_i, C_j) + CCBC(C_i, C_j)}{2}$$

Then, it computes the *cohesion* of each package as the average *coupling* between all pairs of classes contained in it. Indeed, highly cohesive packages should contain highly coupled classes (i.e., classes implementing similar responsibilities). Thus, packages having low level of such quality metrics are identified by ARIES as packages needing restructuring. Once ARIES measures the cohesion of all packages in the system, it highlights those having a cohesion lower than the first quartile of this distribution, as candidates to be "promiscuous".

EPR recommender. This subsystem is in charge of providing suggestions about possible re-modularizations of a package, previously identified as promiscuous. It implements

the approach defined by Bavota *et al.* [6]. This approach takes as input a package identified as a candidate for re-modularization. The package is parsed to build a *class-by-class* matrix, that is a $n \times n$ matrix, where n is the number of classes in the package. An entry $c_{i,j}$ in the matrix represent the probability that classes C_i and C_j should stay in the same package. The value of the entry is computed using a hybrid coupling measure obtained through a weighted average of the two metrics described above (i.e., ICP and CCBC). Once the *class-by-class* matrix is built, its transitive closure is computed to extract chains of strongly related classes. Each chain represents a new package containing classes having similar responsibilities (i.e., highly coupled classes) and that should be in the same package after the refactoring process. However, in the *class-by-class* matrix there could be very few zero values, due to spurious (but light) structural and/or conceptual relationships between classes [7]. Thus, a transitive closure may include almost all the classes in a single chain. To avoid such a problem and to identify the strongest relationships between classes, the *class-by-class* matrix is filtered based on a coupling threshold named *minCoupling*. All values lower than *minCoupling* are converted to zero. A possible side effect of this step is the extraction of short (trivial) chains. In order to avoid the extraction of *trivial* packages, with a very low number of classes in them, the algorithm uses a *minLength* threshold to ensure that the extracted chains contain at least *minLength* classes. Each extracted chain, containing a number of classes lower than *minLength*, is merged with the most similar *non trivial* chain. This parameter, as all the others parameters in the approach by Bavota *et al.* [6] (e.g., *minCoupling* and the weights used when merging the ICP and the CCBC metrics) are customizable in ARIES (see Figure 2).

Note that, if the output of the EPR algorithm is a single chain, then no re-modularization is suggested by the tool. Indeed, this usually happens when the cohesion of the package is very high and it does not need any re-modularization. Otherwise, if the refactoring is accepted by the developer, then ARIES performs the refactoring, modifying the source code as needed to avoid syntactical errors.

Topic Map Manager. This subsystem allows to build the topic map for a generic package *P* by analyzing the term frequencies in the classes it contains. In particular, given a package P, ARIES extracts all the terms in its classes and exploits a stop-word list to prune out common English words and Java keywords[2]. Then, the five most popular terms (i.e., those present in the highest number of classes) are used to construct the topic map of *P* 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 classes in *P* that implements the corresponding topic. The graphical representation of the main topics of *P* is then obtained by tracing lines between the percentage points on each of the five axes, indicating the percentage of classes belonging to *P* that implement the corresponding topic. The topic map provided for the promiscuous package is meant to help the developer in understanding which are the different responsibilities implemented in the package. Clearly not all topic maps will be equally helpful, as they depend on the meaningfulness of identifiers and comments in the code.

[2]The employed stop-word list can be customized in ARIES.

IV. Evaluation

The EPR approach implemented by ARIES has been empirically evaluated in a study involving 16 Master students who analyzed refactoring operations suggested by the approach in order to understand (i) if the extracted packages have a higher quality than the original (promiscuous) one; and (ii) if the extracted packages are meaningful from a functional point of view [6]. The results of the evaluation indicate that the decomposed packages have better cohesion without a deterioration of coupling and the re-modularization proposed by the tool are meaningful from a functional point of view.

V. Demo Remarks

In this demo we presented the Extract Package Refactoring feature of ARIES, a tool that supports developers in the identification of packages that need to be re-organized and the consequent resolution of the design problem through the application of refactoring operations. As future work, we plan to (i) extend the functionalities of the tool to support other refactoring operations, such as, Move Class refactoring; and (ii) implement a more sophisticated approach to detect packages that need to be refactored.

VI. Acknoledgements

Fabio Palomba is partially funded by the University of Molise.

References

[1] M. M. Lehman, "On understanding laws, evolution, and conservation in the large-program life cycle," *J. Syst. Softw.*, vol. 1, pp. 213–221, Sep. 1984. [Online]. Available: http://dx.doi.org/10.1016/0164-1212(79)90022-0

[2] M. Hall, M. Khojaye, N. Walkinshaw, and P. McMinn, "Establishing the source code disruption caused by automated remodularization tools," in *Software Maintenance and Evolution, 2014. ICSME '14. 30th International Conference on*, 2014.

[3] G. Bavota, A. D. Lucia, A. Marcus, R. Oliveto, and F. Palomba, "Supporting extract class refactoring in eclipse: The ARIES project," in *34th International Conference on Software Engineering, ICSE 2012, June 2-9, 2012, Zurich, Switzerland*, 2012, pp. 1419–1422.

[4] Y. S. Lee and B. S. Liang, "Measuring the coupling and cohesion of an object-oriented program based on information flow," in *Software Quality*, 1995.

[5] 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.

[6] G. Bavota, A. De Lucia, A. Marcus, and R. Oliveto, "Using structural and semantic measures to improve software modularization," *Journal of Empirical Software Engineering (EMSE)*, pp. 901–932, 2013. [Online]. Available: http://dx.doi.org/10.1016/0164-1212(79)90022-0

[7] G. Canfora, J. Czeranski, and R. Koschke, "Revisiting the delta ic approach to component recovery," in *Reverse Engineering, 2000. Proceedings. Seventh Working Conference on*, 2000, pp. 140–149.