Empir Software Eng (2013) 18:901–932
DOI 10.1007/s10664-012-9226-8

Using structural and semantic measures to improve software modularization

Gabriele Bavota · Andrea De Lucia · Andrian Marcus · Rocco Oliveto

Published online: 14 September 2012

Editors: Giuliano Antoniol and Martin Pinzger

Abstract Changes during software evolution and poor design decisions often lead to packages that are hard to understand and maintain, because they usually group together classes with unrelated responsibilities. One way to improve such packages is to decompose them into smaller, more cohesive packages. The difficulty lies in the fact that most definitions and interpretations of cohesion are rather vague and the multitude of measures proposed by researchers usually capture only one aspect of cohesion. We propose a new technique for automatic re-modularization of packages, which uses structural and semantic measures to decompose a package into smaller, more cohesive ones. The paper presents the new approach as well as an empirical study, which evaluates the decompositions proposed by the new technique. The results of the evaluation indicate that the decomposed packages have better cohesion without a deterioration of coupling and the re-modularizations proposed by the tool are also meaningful from a functional point of view.

Keywords Software re-modularization · Information-flow-based coupling · Conceptual coupling between classes · Empirical studies

G. Bavota · A. De Lucia
University of Salerno, Fisciano SA, Italy

G. Bavota
e-mail: gbavota@unisa.it

A. De Lucia
e-mail: adelucia@unisa.it

A. Marcus
Wayne State University, Detroit, MI 48202, USA
e-mail: amarcus@wayne.edu

R. Oliveto (✉)
University of Molise, Pesche IS, Italy
e-mail: rocco.oliveto@unimol.it

1 Introduction

During the software development life cycle, changes are inevitable (Lehman 1980). A software system evolves as changes in the environment and requirements are incorporated in the system. Empirical studies have indicated that over 90 % of the costs of a typical system arise in the maintenance phase. A key point for sustainable program maintenance is to tackle software complexity. In Object-Oriented (OO) software, packages group together logically and structurally related classes aiming at 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 the software system changes. 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, software quality decreases in time. In such cases a re-modularization of the system is recommended.

In this paper we focus on a specific restructuring problem in the context of OO software: given a package with poor cohesion, decompose it into smaller and meaningful packages that have higher cohesion. The problem is deceivingly simple; the challenges stem from the definition of cohesion. The concept of software cohesion has been defined by Stevens et al. (1974), who defined it as *the degree to which the internal contents of a module are related*. In OO software, cohesion is usually applied at class level and it can be extended to package level. In our context, classes are the elements of a package and any cohesion measure should reflect how they "belong together". Classes relate to one another in more than one way and previous work (Marcus et al. 2008) has shown that no single metric captures all aspects of cohesion. It is a result that motivates our work. We approach the package re-modularization problem by adapting two class-level coupling metrics (i.e., Information-Flow-based Coupling (ICP) (Lee et al. 1995) and Conceptual Coupling Between Classes (CCBC) (Poshyvanyk et al. 2009)) to measure package cohesion. These measures capture structural and semantic relationships between classes, respectively. The use of the ICP measure allows us to capture the amount of information flowing between the classes of the system via parameters through method invocations. In other words the ICP measure provides information about the structural cohesion of a package, which corresponds to highest cohesion levels of a module (Stevens et al. 1974). On the other side, the CCBC measure captures the lexical information embedded in the comments and identifiers of the classes, allowing to identify semantically (i.e., domain semantics) related classes, i.e., classes containing similar terms in their comments and identifiers. Thus, the CCBC captures lower cohesion levels in a module (such as logical cohesion (Stevens et al. 1974)), concerned with cases of classes that deal with common concepts and that make sense to package together. We combine these two sources of information and use an aggregated measure to determine classes that should belong together in a package. The technique is automated and suggests to developers how to split existing packages, when needed.

A first version of the proposed approach together with a preliminary evaluation on three software systems has been previously presented (Bavota et al. 2010a).

In this paper we extend the approach and its empirical evaluation. The specific contributions of this paper can be summarized as follows:

- an analysis of the impact of the algorithm's configuration parameters on the performances of the proposed approach. The analysis allowed us to derive an heuristic based on Principal Component Analysis (PCA) to identify of default values for the configuration parameters. The proposed heuristic improves the usability of the proposed approach by facilitating its application in real usage scenarios;
- an analysis of the orthogonality of the measures used to capture relationships between classes. We also empirically analyze the role of the semantic measures in the context of software re-modularization;
- the replication of the preliminary evaluation. In addition to the three systems used in Bavota et al. (2010a), the evaluation has been extended to include two other software systems. In the context of the study we merged several packages of the object systems and used the proposed approach to split the merged package aiming at reconstructing the original packages. Our assumption is that the higher the reconstruction accuracy of our approach, the higher the meaningfulness of the proposed re-modularization. This assumption is supported in part by our choice of systems, which have high quality. However, to further verify this assumption in the cases where the re-modularization proposed by our approach is significantly different from the original decomposition of the system, we asked some of the original developers of the object systems to analyze the proposed package decompositions and evaluate them from a functional point of view.

The last contribution, i.e., the analysis of the meaningfulness of the recommended re-modularizations, is a fundamental issue. In a recent survey by Praditwong et al. (2011) was found that there is no literature in software re-modularization that evaluated from the point of view of the software engineer, the modularizations suggested by the tools. This is an important factor given the vague definition of cohesion in general and the fact that most cohesion measures capture only certain aspects of cohesion (e.g., common attribute references). In addition, modules with poor cohesion are either poorly constructed or (more often) they implement different and unrelated functionalities. Most metrics do not distinguish between these two causes of poor cohesion, hence, once again, it is important to evaluate such approaches from a functional point of view.

The rest of the paper is organized as follows. Section 2 discusses the related research, while Section 3 presents the proposed approach. Sections 4 and 5 present the design and the results of the empirical study, respectively. Section 6 discusses lessons learned and threats to validity that could affect the results of our empirical study. Finally, Section 7 gives concluding remarks and presents future work.

2 Related Work

A lot of effort has been devoted to the definition of automatic and semi-automatic approaches aimed at supporting software engineers in the re-modularization of software systems. Since the 80's, many authors investigated how to increase the quality of procedural programs, in terms of maintainability, reusability, and high

level design, by restructuring the software architecture. In particular, many approaches have been proposed to aggregate procedures with high functional cohesion (Antoniol et al. 2001; Cimitile and Visaggio 1995; Shaw et al. 2003). Most of these approaches are based on identifying strongly connected sub-graphs in the call graph representing the program. Cimitile and Visaggio (1995) proposed a technique based on dominance trees to aggregate procedures in reusable modules. An improvement of this technique has been proposed in Shaw et al. (2003) to support program comprehension. Antoniol et al. (2001) proposed the use of concept analysis to restructure the architectural organization of the source code files in legacy systems. Other techniques have been proposed for the identification of objects or ADTs (Abstract Data Types) in legacy systems. Such approaches generally identify objects or ADTs in legacy code exploiting the relations existing between program routines and global variables and/or user defined data types (see e.g., Canfora et al. 2001; van Deursen and Kuipers 1999; Koschke et al. 2006; Tonella 2001).

Most of the work on re-modularization is based on clustering techniques. Wiggerts (1997) provides the theoretical background for the application of cluster analysis in software re-modularization. The paper discusses how to establish similarity criteria between the entities to cluster and gives a summary of possible clustering algorithms to use in software re-modularization. Anquetil and Lethbridge (1999) tested some of the algorithms proposed by Wiggerts and compared their strengths and weaknesses when applied to system re-modularization. They performed this study on several software systems, including a real world legacy system. Mitchell and Mancoridis (2001) proposed some guidelines to compare the performance of different clustering algorithms for source code decomposition, while a more recent work by Shtern and Tzerpos (2009) introduced a method for the selection of a suitable clustering algorithm, given specific needs. Wu et al. (2005) conducted a comparative study of clustering algorithms in the context of software evolution. In particular, the authors focused their attention on the stability of clustering algorithms, i.e., a clustering algorithm is stable if it produces very similar output given very similar input, like two subsequent versions of the same system with no major changes. Their results show that for large systems the analyzed clustering algorithms are not ready to be widely adopted. Maqbool and Babri (2007) focused on the application of hierarchical clustering in the context of software architecture recovery and modularization. They investigated the measures to be used in this domain, categorizing various similarity and distance measures into families according to their characteristics. The behavior of various clustering algorithms was also studied on four large legacy systems.

Mancoridis et al. (1998) proposed an automatic technique aiming at creating a high-level view of the system organization. In particular, they introduced a search-based approach that uses a hill-climbing based clustering to identify the organization of a software system. The same technique is also used in Mitchell and Mancoridis (2006) where the authors present Bunch, a tool supporting automatic system decomposition. Search-based approaches are also used in Abdeen et al. (2009), Harman et al. (2002) and Seng et al. (2005). In particular, in Harman et al. (2002) and Seng et al. (2005) the authors used a genetic algorithm to improve the subsystem decomposition of a software system. The fitness function to be maximized is defined using a combination of quality metrics, e.g., coupling, cohesion, and complexity. Abdeen et al. (2009) proposed a heuristic search-based approach for automatically optimizing, i.e. reducing, the dependencies between packages of a software system.

Starting from an initial decomposition, their technique optimizes the existing package structure by moving classes between the original packages.

The combined use of semantic and structural measures for software re-modularization is the main characteristic of our approach. The re-modularization approaches discussed above exploit only information derived by structural metrics. A lot of important information is embedded in the comments and identifiers of the system's classes. From this point of view our approach is closer to Corazza et al. (2010, 2011), Maletic and Marcus (2001) and Scanniello et al. (2010). In particular, Maletic and Marcus (2001) combined semantic and structural measures to identify ADTs in legacy code. They used Latent Semantic Indexing (LSI) (Deerwester et al. 1990), an Information Retrieval (IR) technique, to capture semantic relationships between source artifacts. The aim of our approach is different: we combine structural and semantic class coupling measures to decompose a package with low cohesion in two or more packages with much higher cohesion. Kuhn et al. (2007) also used LSI to cluster together source artifacts that use a similar vocabulary. Moreover, they provided a visual notation that gives an overview of all the clusters and their semantic relationships. Their approach is focused on the identification of topics in the source code and, for this reason, only uses semantic information.

Ducasse et al. (2007) introduced the Package Surface Blueprint, a visual approach for understanding package relationships in complex software systems. In particular, the authors showed that their approach helps users in identifying poorly designed packages.

Corazza et al. (2010, 2011) presented a clustering based approach to partition object oriented systems into subsystems. In particular, they extracted lexical information from the source code and use the K-Medoids partitioning algorithm (Corazza et al. 2010) or the Hierarchical Agglomerative Clustering algorithm (Corazza et al. 2011) to build subsystems containing semantically related classes. Unlike our approach, in Corazza et al. (2010, 2011) the structural dependencies between the classes are ignored. Moreover, the number of clusters, i.e., subsystems, to build must be fixed a priori (in Corazza et al. (2010) it is arbitrarily set to the 10 % of the number of classes, yet this value is not empirically evaluated). Conversely, our approach is able to automatically identify the number of packages in which a set of classes should be organized.

A clustering based approach using both structural and lexical information is also proposed by Scanniello et al. (2010), but it is focused on recovering a layered architecture from the source code of object oriented systems. In particular, the structural information is used by the Kleinberg algorithm (Kleinberg 1999) to identify software layers, while lexical information is employed to partition each identified layer into software modules using the k-means algorithm (Hartigan 1975). In a related research thread, semantic clustering was also recently used to support the comprehension of web sites (Ricca et al. 2008).

The focus of our approach is different with respect to the aforementioned works. Most of the approaches in literature (see, e.g., Mancoridis et al. 1998; Shtern and Tzerpos 2009; Wiggerts 1997; Wu et al. 2005) support the software engineer during a "big-bang" re-modularization, i.e., all the classes of the software system are reorganized in a new package decomposition. Our approach is different: we aim at incrementally re-modularizing a software system focusing on a single package at a time.

3 The Proposed Approach

The proposed approach analyzes the (structural and semantic) relationships between classes in a package to suggest a possible re-modularization in two or more packages. The combined use of structural and semantic coupling metrics allows us to analyze the package's cohesion from the point of view of the dependencies between the classes and the point of view of the responsibilities of the classes from the package. Generally, a package with loosely coupled classes both from a structural and semantic point of view exhibits low cohesion. Thus, extracting new packages from the original package may improve its cohesion.

The proposed approach takes as input a package identified by the software engineer as a candidate for re-modularization. Then, a measure reflecting a relationship between pairs of classes from 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 $m_{i,j}$ in the class-by-class matrix represents the likelihood that class c_i and class c_j should be in the same package.

Using the information in the class-by-class matrix the approach extracts chains of strongly related classes. The classes of the original package are distributed in different packages according to the extracted chains. If the number of extracted chains is one, no re-modularization is suggested by the tool (this generally happens when the cohesion of the analyzed package is high). Otherwise, based on the extracted class chains the approach suggests new packages with higher cohesion than the original package. Note that the structure of individual classes in the package is not changed.

While the proposed approach is automated, it is actually supposed to serve as an assistant to the developer. Design decisions are often more complex and subtle than just trying to maximize package cohesion. In consequence, the extracted packages are analyzed by the software engineer who can accept the proposed re-modularization as is, or change it by moving classes from one package to another.

3.1 Class-by-Class Matrix Construction

The likelihood that class c_i and class c_j should be in the same package is estimated by capturing different types of relationships between classes that can affect package cohesion. In the context of this work, we define the likelihood that two classes should be in the same package by combining two different (structural and semantic) measures, i.e., information-flow-based coupling (ICP) (Lee et al. 1995) and Conceptual Coupling Between Classes (CCBC) (Poshyvanyk et al. 2009).

Our choice of metrics to use is not random, as it is based on previous research (Poshyvanyk et al. 2009) that analyzed the combination of structural and semantic coupling metrics to predict changes in OO software. ICP and CCBC fared better than other structural and conceptual metrics, respectively. Moreover, the empirical analysis conducted in Poshyvanyk et al. (2009) have shown that structural and semantic coupling measures do not correlate, which indicates that they capture different aspects of coupling. In light of these results, we expect that (i) a package composition based on these metrics will group together classes that tend to change together, which is a desirable property in order to localize change, and (ii) the use of

a combination of orthogonal quality metrics to guide the re-modularization activity can provide better results than any one of its constituents (Maletic and Marcus 2001; De Lucia et al. 2008). However, the choice of coupling metrics may have a strong impact on the performances of a re-modularization method (Harman et al. 2005) and therefore, as future work we plan to investigate other combinations of metrics.

The CCBC measure is based on the semantic information (i.e., domain semantics) captured in the code by comments and identifiers. Two classes are conceptually related if their (domain) semantics are similar, i.e., they have similar responsibilities. The definition of CCBC requires the introduction of a lower level measure (Poshyvanyk et al. 2009): the Conceptual Coupling Between Methods (CCM).

To measure CCM, Latent Semantic Indexing (LSI) is used to represent each method as a real-valued vector that spans a space defined by the vocabulary extracted from the code. The conceptual coupling between two methods m_i and m_j is then calculated as the cosine of the angle between their corresponding vectors (Baeza-Yates and Ribeiro-Neto 1999):

$$CCM(m_i, m_j) = \frac{\overrightarrow{m_i} \cdot \overrightarrow{m_j}}{\|\overrightarrow{m_i}\| \cdot \|\overrightarrow{m_j}\|}$$

where $\overrightarrow{m_i}$ and $\overrightarrow{m_j}$ are the vectors corresponding to the methods m_i and m_j, respectively, and $\| \overrightarrow{x} \|$ represents the Euclidean norm of the vector x (Baeza-Yates and Ribeiro-Neto 1999). Thus, the higher the value of CCM the higher the similarity between two methods. It is clear that CCM depends on the consistency of naming used in the source code and comments. Note that other IR methods could be used here, as well as other similarity measures.

Now we can define the conceptual coupling between two classes c_i and c_j as:

$$CCBC(c_i, c_j) = \frac{\sum_{m_h \in c_i} \sum_{m_k \in c_j} CCM(m_h, m_k)}{|c_i| \times |c_j|}$$

where $|c_i|$ ($|c_j|$) is the number of methods in c_i (c_j). Thus, $CCBC(c_i, c_j)$ is the average of the coupling between all unordered pairs of methods from class c_i and class c_j. The definition of this measure ensures that CCBC is symmetrical, i.e., $CCBC(c_i, c_j) = CCBC(c_j, c_i)$.

Concerning the structural information, the ICP measure is based on method invocations between classes. In particular, ICP measures the amount of information flowing into and out of a class via parameters through method invocation, i.e., the measure sums the number of parameters passed at each method invocation. Note that the ICP metric also implicitly captures class references performed through method calls, such as, the invocation of a constructor when an instance variable is created. Like the majority of coupling metrics in literature, this metric is defined at the system level, i.e., for a given class c all method calls between c and *all* other classes in the system are taken into account. For our approach we need to redefine ICP to take into account coupling between a pair of classes. We use the ICP metric as redefined in Poshyvanyk et al. (2009); the information-flow-based coupling between a pair of 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,[1] weighted by the number of parameters of the invoked methods:

$$ICP_{i\to j} = \sum_{k=1}^{|calls(c_i,c_j)|} p(call(c_i, c_j)_k)$$

where $p(call(c_i, c_j)_k)$ is the number of parameters in the $k-th$ call from c_i to c_j.

To ensure that ICP represents a commutative measure we define the overall information-flow-based coupling between the classes c_i and c_j as follows:

$$ICP(c_i, c_j) = ICP(c_j, c_i) = \max\left\{ICP_{i\to j}, ICP_{j\to i}\right\}$$

While CCBC has value in [0, 1], ICP is equal or higher than 0 but it is unbounded on top. To combine the two metrics in a single similarity measure we need to normalize the ICP values in [0, 1]. To this aim, we defined the normalized ICP as follow:

$$\widetilde{ICP}(c_i, c_j) = \frac{ICP(c_i, c_j) - \min_{ICP}}{\max_{ICP} - \min_{ICP}}$$

where $\min_{ICP}$ ($\max_{ICP}$) is the minimum (maximum) $ICP(c_i, c_j)$ value measured between the classes of the package to re-modularize.

Finally, we compute the likelihood that classes c_i and c_j should be in the same package as:

$$coupling_{i,j} = w_{ICP} \cdot \widetilde{ICP}(c_i, c_j) + w_{CCBC} \cdot CCBC(c_i, c_j)$$

where $w_{ICP} + w_{CCBC} = 1$ and their values express the confidence (i.e., weight) in each measure. The weights assigned to these measures are empirically defined and the methodology for this step is presented in Section 5.

3.2 Class Chains Extraction

The extraction of the class chains is performed in *two steps*. In the *first step* we obtain a set of chains based on the transitive closure of the class-by-class matrix. However, in the class-by-class matrix there could be very few zero values, due to spurious (but light) structural and/or semantic relationships between classes (Koschke et al. 2006). 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, we filter the class-by-class matrix, based on a coupling threshold. All values less than a threshold, *minCoupling*, are converted to zero:

$$\widetilde{m}_{i,j} = \begin{cases} coupling_{i,j} & \text{if } coupling_{i,j} > minCoupling; \\ 0 & \text{otherwise.} \end{cases}$$

In this way we capture only the strongest relationships in the class-by-class matrix, ensuring that spurious structural and/or semantic relationships between classes are ignored (Koschke et al. 2006). There are many ways to define a threshold aimed

[1]The method call are captured through static analysis of the source code.

at removing spurious relationships between classes. A simple classification allow to identify two different kinds of thresholds:

- *constant threshold*: the value of the threshold is fixed *a priori*, e.g., $minCoupling = 0.1$. This kind of threshold is simple to implement, but in general it is very difficult to choose *a priori* a constant value to prune-out spurious relationships. Indeed, the values in the class-by-class matrix depend on the package chosen to be refactored. In fact, there may be cases where the matrix contains a lot of high values. In this case, if the fixed threshold is high, it will probably remove the noise from the matrix, e.g., spurious relationships between the classes of the package. Otherwise almost all the values will be left in the matrix. On the other hand, there may be cases where the matrix contains a high number of very low values. In this case, a high constant threshold will remove almost all the information from the matrix.
- *variable threshold*: the value of the threshold is automatically selected taking into account the characteristics of the given input. For example, *minCoupling* can be set as the median of the values present in the class-by-class matrix. This kind of threshold should resolve the problems derived by the use of a *constant threshold* and should ensure more stable filtering performances across the different inputs. Choosing the best threshold in this case is far from trivial.

We experimented with both *constant* and *variable* thresholds to empirically validate the above observations. The findings allowed to define a heuristic to select the best threshold (see Section 5 for details).

Once the *first step* is completed (i.e., the transitive closure of the class-by-class matrix is computed), it is possible that the set of computed chains (i.e., suggested packages) may include chains with a very short length due to classes having poor relationships with other classes. In the *second step*, to avoid suggesting very small packages (i.e., packages with a very low number of classes), we use a chain length threshold, *minLength*, to identify trivial chains, i.e., chains with a length less than *minLength*. Similar to Bittencourt and Guerrero (2009), in our approach we set $minLength = 4$ since a good re-modularization approach should avoid the creation of packages with too few classes. This minimum length can be easily changed if needed. Then, we compute the coupling between trivial and non-trivial chains and merge each trivial chain with the strongest coupled non-trivial chain. The coupling between chains is calculated using the same measures used to calculate the coupling between classes. Specifically, the coupling between chains Ch_i and Ch_j is computed as the average coupling between all the possible pairs of classes from Ch_i and Ch_j:

$$Coupling(Ch_i, Ch_j) = \frac{\sum_{c_i \in Ch_i} \sum_{c_j \in Ch_j} coupling_{i,j}}{|Ch_i| \times |Ch_j|}$$

where $|Ch_i|$ ($|Ch_j|$) is the number of classes in Ch_i (Ch_j).

4 Case Study Design

In this section we describe in detail the design of the case study we carried out to empirically assess the proposed re-modularization technique and the results we obtained. The study follows the Goal-Question-Metrics paradigm (Basili et al. 1994).

Table 1 Systems used in the case study

System	Version	KLOC	#Classes	#Packages	Cohesion		
					Mean	Median	St. dev
eTour	1.0	30	134	17	0.348	0.311	0.067
GESA	2.0	46	297	22	0.332	0.289	0.120
JHotDraw	6.0 b1	29	275	12	0.364	0.379	0.052
SESA	1.2	11	128	14	0.318	0.292	0.073
SMOS	1.0	23	121	12	0.400	0.424	0.039

4.1 Definition and Context

The *goal* of the empirical study is (i) to assess the parameters of the proposed approach, i.e., the weights of the coupling metrics (w_{ICP}, and w_{CCBC}) and the *class-by-class matrix* filtering threshold (*minCoupling*), and (ii) to determine whether the proposed approach generates meaningful re-modularization of packages in object-oriented software systems. The *quality focus* is ensuring a better package quality, while the *perspective* is both (i) of a researcher, who wants to evaluate how the combination of structural and semantic similarity measures between classes can support software re-modularization; and (ii) of a project manager, who wants to evaluate the possibility of adopting the proposed technique within his/her software company.

The objects of our study are: an open source system, JHotDraw[2] and four software systems (eTour, GESA, SESA, and SMOS) developed by university students during a Software Engineering course. JHotDraw is a Java GUI framework for structured drawing editors. eTour is an electronic touristic guide, while GESA is a web-based application used in the management of university courses. SESA is also a web-based application used to manage relevant information of the Software Engineering Lab of the University of Salerno, e.g., people, projects, publications. Finally, SMOS is a software developed for high schools, which offers a set of functionalities aimed at simplifying the communications between the school and the students' parents. Table 1 reports the statistics (i.e., KLOC, number of classes, and number of packages) as well as the versions of the systems used in the study. The table also reports the descriptive statistics of the packages' cohesion from the systems, measured using a cohesion metric, namely *CohesionQ*, defined in Abdeen et al. (2009). Our evaluation strategy requires that the object systems have high package cohesion. The measures support our choice, as the average cohesion values for the five systems are higher than that of the systems analyzed in Abdeen et al. (2009): JEdit (with average cohesion 0.288), ArgoUML (0.172), Jboss (0.125), and Azureus (0.117).

4.2 Research Questions and Planning

By construction, the approach will extract from a package, class chains having higher cohesion than the original package. However, as we mentioned before a good re-modularization cannot be based only on the higher cohesion of the new packages.

[2] http://www.jhotdraw.org

An evaluation involving developers is required in order to assess the overall quality and meaningfulness of the proposed re-modularization. For this reason, our study aims at (i) assessing the parameters of our approach and (ii) analyzing if the proposed approach is able to identify meaningful re-modularization operations from a developer point of view. Thus, two research questions are formulated:

– RQ_1: How do the parameters of the proposed approach affect the results?
– RQ_2: Is the proposed approach able to find meaningful re-modularizations?

To respond to our research questions we mutated the original version of the object systems using a tool that randomly selects $m \geq 2$ packages of the system and merges them in a single package $\widehat{P}_m$. The merging operation was recorded in a log file to allow us to know the packages merged by the tool. At the end of the merging operation we obtained a mutated system with a worse package decomposition compared to the original system. The proposed approach is then applied to the $\widehat{P}_m$ package in order to reconstruct (or improve) the original packages. At the end of the re-modularization operation we obtained a new version of the mutated system. Specifically, given the merged package $\widehat{P}_m$, the proposed approach is expected to generate m packages. To evaluate the proposed approach, the new packages were compared with the originally merged packages aiming at identifying the total number of classes correctly and incorrectly placed in the new packages. The ideal behavior is that the split packages are the same (i.e., contain the same classes) as the original packages. In essence, we consider them as a "golden standard". This choice is supported by the fact that the systems used in the study have a generally good package quality that is reflected in terms of package cohesion (see Table 1). In particular, one of the object systems, JHotDraw, has been developed as a "design exercise" and its design relies heavily on using well-known design patterns. The other four systems were chosen among the best projects developed during the software engineering course. As shown in Table 1 the cohesion of the packages of the other four systems is close to that of JHotDraw. With that in mind, recovering the original packages likely means that the approach is able to identify meaningful re-modularizations.

In order to respond to our first research question, we identified different re-modularization solutions on the same merged packages, i.e., mutated systems, using different settings of *weights* for the selected metrics, i.e., w_{ICP}, and w_{CCBC}, and different values for the *threshold* used to remove spurious relationships from the class-by-class matrix, i.e., the parameter *minCoupling*. In particular, for each metric weight we varied this parameter starting at 0 and increasing it until 1 by a step of 0.1. We exercised all the possible combinations of such values assuring that $w_{ICP} + w_{CCBC} = 1$, i.e., 11 different combinations. Concerning the parameter min*Cohesion* we experimented the two types of thresholds described in Section 3.2, i.e., *constant* and *variable* threshold. In particular, we used four different *constant thresholds* and three different *variable thresholds*. Concerning the *constant thresholds* we used 0.1, 0.2, 0.3, 0.4, and 0.5, while as *variable thresholds* we considered the first (Q_1), the second (Q_2), and the third (Q_3) quartile, respectively, of the non-zero values in the class-by-class matrix. Note that the use of quartiles allows to define a threshold that is less affected—as compared to the other descriptive statistics (e.g., mean)—by problems caused by skewed distributions of the values in the class-by-class matrix. We selected different values for the number of packages to be merged, i.e., $m \in \{2, 3, 5\}$, aiming at obtaining merged packages with a low cohesion and

varied set of responsibilities. For each value of m we performed 10 different trials, i.e., $n = 10$, randomly selecting each time different combinations of the merged packages. In total, we did 30 merging and re-modularizations operations for each system (varying on the 11 combinations of weights, i.e., w_{ICP} and w_{CCBS}, and on 8 different values for *minCoupling*). Thus, the total number of trials performed on each object system is $11 \times 8 \times 30 = 2,640$, for a total of 13,200 re-modularizations for the five systems.

To evaluate the results produced by the configurations experimented for our approach, we used two well-known Information Retrieval (IR) metrics, namely recall and precision (Baeza-Yates and Ribeiro-Neto 1999):

$$recall_{P_i} = \frac{|C(\widehat{P}_i) \cap C(P_i)|}{|C(P_i)|} \qquad precision_{P_i} = \frac{|C(\widehat{P}_i) \cap C(P_i)|}{|C(\widehat{P}_i)|}$$

where P_i and $\widehat{P}_i$ are the original and the reconstructed packages, respectively, while $C(P_i)$ is the set of classes of P_i. Recall measures the percentage of classes correctly placed in the split packages, while precision measures the percentage of classes that are correctly placed. Since the two metrics measure two different concepts, a balance between them is usually measured using an aggregate metric, namely the F-measure (Baeza-Yates and Ribeiro-Neto 1999), which is the harmonic mean of the precision and recall:

$$F\text{-}measure_{P_i} = 2 * \frac{precision_{P_i} * recall_{P_i}}{precision_{P_i} + recall_{P_i}}$$

We decided to use the *F-measure* as the dependent variable to assess the performances of the proposed approach and to guide the selection of the best values for the weights and parameters described above. Note that the F-measure is computed analyzing only the reconstruction of the merged packages and not the entire system decomposition.

The reconstruction accuracy (F-measure) achieved using the best parameters setting is also used to respond to our second research question. Our assumption is that the higher the reconstruction accuracy of our approach, the higher the meaningfulness of the proposed re-modularization. As explained before, this assumption is supported in part by our choice of systems which have a high quality in terms of package cohesion. In order to further verify this assumption (and implicitly answer the second research question) we analyzed the proposed re-modularization operations from a functional point of view. In particular, in the cases where the re-modularization proposed by our approach is considerably different from the original decomposition of the system, we asked some of the original developers of eTour, GESA, SESA, and SMOS to analyze the proposed package decomposition and evaluate the performed re-modularizations. For JHotDraw, the same evaluation was made by two graduate students who are very familiar with the system. We involved a total of 16 subjects in the functional evaluation distributed among the object systems as reported in Table 2. To identify the cases to analyze, we set an F-measure threshold ϵ. All the cases for which our approach was not able to reconstruct the original packages with an F-measure higher than ϵ were analyzed by the students. For each of the selected cases the students responded to the following question:

Is the proposed package decomposition meaningful?

Table 2 Subjects involved in the functional evaluation

System	#Subjects	Original developers?
eTour	2	Yes
GESA	5	Yes
JHotDraw	2	No
SESA	2	Yes
SMOS	5	Yes

with a score using a 5-point Likert scale (Oppenheim 1992): 1: *Strongly agree*; 2: *Weakly agree*; 3: *Neutral*; 4: *Weakly disagree*; 5: *Strongly disagree*.

5 Empirical Study Results

In this section we present the results of the empirical study. We discuss two aspects of the results. First, we use the results to determine the best values for the weights and parameters. Second, we analyze the results of the students' evaluation of the proposed re-modularizations.

Table 3 reports the results produced—in terms of *F-measure*—by the *best* configuration of parameters identified on each of the object systems.[3] The results in Table 3 highlight:

- *the benefits of the second step of our approach*. After merging each trivial chain (i.e., a chain composed of less than 4 classes), with the most similar non-trivial chain (see Section 3.2), we obtained an average increment of the F-measure by about 7 % (with respect to the previous step of the approach);
- *the decrease of the reconstruction accuracy when increasing the number of merged packages.* Indeed, the average F-measure decreases from 88 % when merging 2 packages, to 75 % when merging 3 packages, until 68 % when merging 5 packages;
- *a general rule for setting the parameters of our approach.* The results reveals that the best performances can be obtained using as threshold the third quartile, i.e., Q_3, of the non-zero values of the class-by-class matrix and setting $w_{CCBC} \geq 0.7$. However, the best configuration of weights is slightly different among the object systems. Thus we need to investigate deeper the influence of the configuration parameters on the performances of the proposed approach.

5.1 Influence of the Parameters

To better analyze the influence of the configuration parameters Figs. 1, 2, and 3 show the interaction plots between *Weights* and *Threshold* on GESA, merging 2, 3, and 5 packages, respectively. We report the results obtained using all the three variable thresholds but for sake of readability we only report the results achieved using the lower, the higher, and the best constant threshold (see Bavota et al. (2011a) for the complete interaction plots of all the systems).

[3]The complete results achieved with all the possible combinations of parameters can be found in Bavota et al. (2011a).

Table 3 Descriptive statistics of results achieved reconstructing merged packages

System	m	Best configuration			F-Measure (after step 1)			F-Measure (after step 2)		
		w_{CCBC}	w_{ICP}	Threshold	Mean	Median	Std.dev.	Mean	Median	Std.dev.
eTour	2	0.9	0.1	Q_3	0.804	0.852	0.094	0.891	0.936	0.086
	3	0.9	0.1	Q_3	0.688	0.703	0.102	0.760	0.774	0.082
	5	0.9	0.1	Q_3	0.559	0.562	0.084	0.668	0.659	0.057
GESA	2	0.9	0.1	Q_3	0.917	0.936	0.039	0.967	0.981	0.044
	3	0.9	0.1	Q_3	0.763	0.795	0.114	0.822	0.897	0.131
	5	0.9	0.1	Q_3	0.603	0.578	0.104	0.720	0.706	0.073
JHotDraw	2	0.7	0.3	Q_3	0.749	0.788	0.142	0.785	0.807	0.180
	3	0.9	0.1	Q_3	0.672	0.709	0.100	0.724	0.760	0.085
	5	0.8	0.2	Q_3	0.593	0.635	0.056	0.688	0.675	0.027
SESA	2	0.8	0.2	Q_3	0.897	1.000	0.095	0.930	1.000	0.108
	3	0.8	0.2	Q_3	0.647	0.602	0.125	0.700	0.643	0.109
	5	0.8	0.2	Q_3	0.548	0.522	0.138	0.660	0.600	0.104
SMOS	2	0.8	0.2	Q_3	0.769	0.846	0.206	0.804	0.920	0.263
	3	0.8	0.2	Q_3	0.705	0.720	0.108	0.770	0.790	0.101
	5	0.9	0.1	Q_3	0.558	0.604	0.126	0.668	0.700	0.138

The number of merged packages is m

The analysis indicates that the *variable thresholds* ensure better filtering performances than the constant thresholds across the different inputs, i.e., the different artificial packages to be re-modularized. In particular, the best performances are achieved using Q_3 as threshold to remove spurious relationships in the class-by-class matrix.

Regarding the weights, the results reveal that the weight for the semantic metric, i.e., w_{CCBC}, should be higher than 0.6. In fact, all combination of weights having

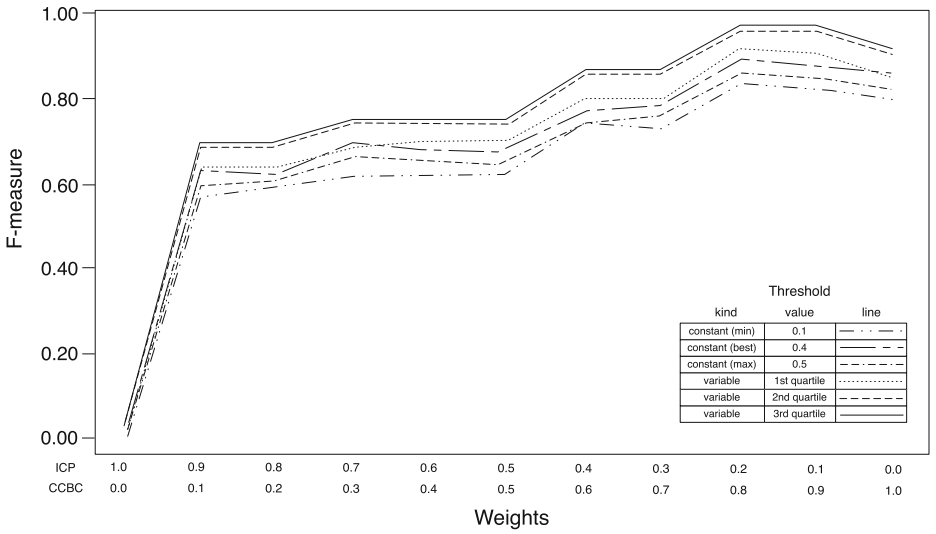

Fig. 1 Interaction between weight and threshold on GESA merging 2 packages

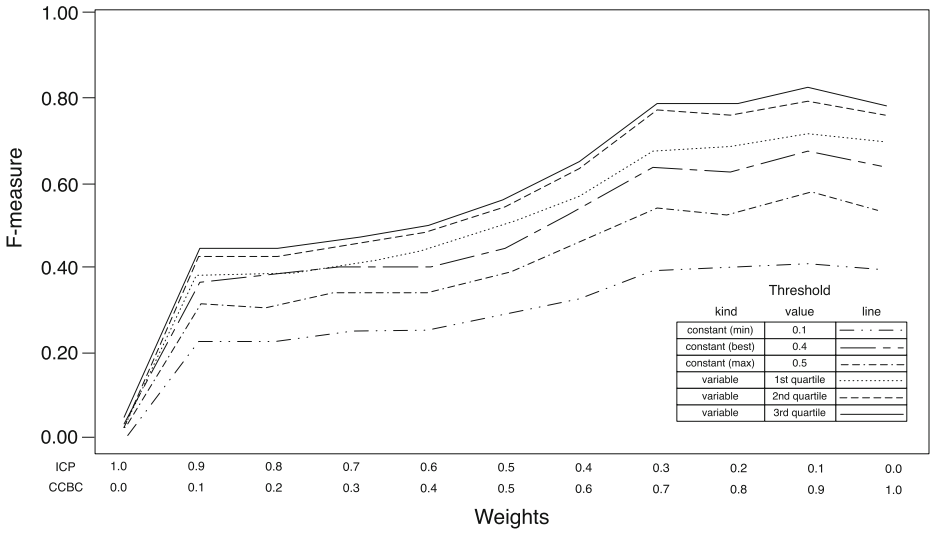

Fig. 2 Interaction between weight and threshold on GESA merging 3 packages

$w_{CCBC} \geq 0.7$ has a reconstruction accuracy almost equal to the best (see Figs. 1, 2, and 3). This trend (as well as the trend concerning the variable threshold Q_3) is confirmed across all the experimented systems (see Bavota et al. 2011a). The high importance (i.e., weight) of the semantic metric probably derives from the fact that even for packages with good structural cohesion there might be pairs of classes with no structural interaction, e.g., two classes with no method calls between each other.

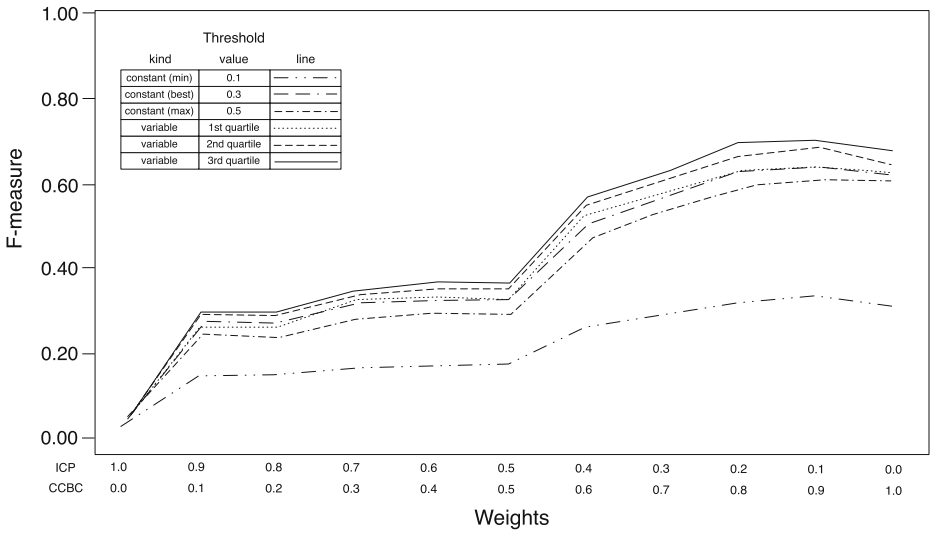

Fig. 3 Interaction between weight and threshold on GESA merging 5 packages

Note that the method calls that we capture are just a subset of all possible ways in which two classes can be structurally related. We extract method calls statically with a rather simple and conservative analysis of the code using Eclipse's AST parser. A more sophisticated analysis would likely yield additional structural relations, which may increase the weight of the structural component of the combined measure. In the cases where there are no structural relationships between classes, only the semantic metric can help to cluster together these pairs of classes, when needed. This fact is also highlighted by the strong performances decrease affecting our approach when the weight for the semantic metric is equal to zero. It is worth noting that even if our approach is really stable across all the configurations of weights having $w_{CCBC} \geq 0.7$, it shows slight decrease of performances when the structural metric, i.e., ICP, is set to zero (see Figs. 1, 2, and 3). All these observations suggest that the semantic metric captures most of the coupling (relevant to our task) between the classes of the object systems and consequently helps to better cluster together classes of the same original package, i.e., classes with high coupling. To verify such a conjecture we apply PCA to the coupling measures. This allows identifying the different dimensions that describe a phenomenon, e.g., the coupling between pairs of classes, and obtain an indication of the importance of each dimension (captured by one or more coupling metrics) in the description of this phenomenon, i.e., the proportion of variance. It is worth noting that as it is defined, the structural metric ICP gives a value different than zero only in case two classes are related at least by one method call. The problem is that the pair of classes that are related through a method call represents only a small percentage of the possible pairs of classes in a software system, e.g., in JHotDraw "only" 2 % out of about 38,000 possible pairs of classes have method calls between them (about 650). In such a situation, the output of the PCA is trivial since it assigns almost all the description of the observed phenomenon, i.e., coupling between classes, to the semantic metric. To avoid this problem, we executed the PCA only on the pairs of classes related trough at least a method call. Table 4 shows the achieved results. As we can see, the semantic metric is identified by the PCA as the metric that describes most of the coupling between pairs of classes (its proportion of variance is always higher than 0.6). Moreover, the proportion of variance values provided by the PCA are close to the weights used for the coupling metrics in the configurations with the best results of the F-measure. Thus, we conjecture that the PCA could be used to weight the exploited coupling metrics taking into account the portion of coupling captured by each metric (proportion of variance). In particular, the higher the proportion of variance captured by one metric, the higher its weight. To verify the usefulness of the PCA in setting the weights for the metrics exploited by our approach we formulated an additional research question:

– RQ_3: Can the proportion of variance obtained by PCA be used to weight the coupling metrics exploited in our approach?

To respond to this research question, we compared the results obtained using as configuration parameters the one identified by the proportion of variance of the PCA, i.e., *PCA-based configuration*, with the best results obtained in our experimentation, i.e., *best configuration*. Table 5 reports the achieved results. As we can see the difference between the reconstruction accuracy of the *PCA-based configuration* compared with the accuracy obtained using the best configuration is very small, i.e., the difference of F-measure is never higher than 0.04. This result indicates that the

Table 4 Results of PCA: rotated components

	PC1	PC2
(a) eTour		
Proportion of variance	0.62	0.38
Cumulative proportion	0.62	1.00
CCBC	0.96	0.26
ICP	0.26	−0.96
(b) GESA		
Proportion of variance	0.90	0.10
Cumulative proportion	0.90	1.00
CCBC	0.99	−0.05
ICP	0.05	0.99
(c) JHotDraw		
Proportion of variance	0.82	0.18
Cumulative proportion	0.82	1.00
CCBC	−0.99	0.07
ICP	−0.07	−0.99
(d) SESA		
Proportion of variance	0.77	0.23
Cumulative proportion	0.77	1.00
CCBC	−0.98	0.21
ICP	−0.21	−0.98
(e) SMOS		
Proportion of variance	0.94	0.06
Cumulative proportion	0.94	1.00
CCBC	−0.99	−0.01
ICP	−0.01	0.99

PCA-based configuration provides an accuracy similar to the best accuracy obtained exploiting all possible configurations. Such results support our conjecture indicating that the proportion of variance provided by the PCA can be used to weight the corresponding metric.

Table 5 Results reconstructing merged classes: PCA based *vs* best configuration

System	# Merged classes	Best configuration	PCA-based configuration
eTour	2	$w_{CCBC} = 0.9, w_{ICP} = 0.1$ (**0.89**)	$w_{CCBC} = 0.6, w_{ICP} = 0.4$ (**0.85**)
	3	$w_{CCBC} = 0.9, w_{ICP} = 0.1$ (**0.76**)	$w_{CCBC} = 0.6, w_{ICP} = 0.4$ (**0.74**)
	5	$w_{CCBC} = 0.9, w_{ICP} = 0.1$ (**0.67**)	$w_{CCBC} = 0.6, w_{ICP} = 0.4$ (**0.63**)
GESA	2	$w_{CCBC} = 0.9, w_{ICP} = 0.1$ (**0.97**)	same as the best configuration
	3	$w_{CCBC} = 0.9, w_{ICP} = 0.1$ (**0.82**)	same as the best configuration
	5	$w_{CCBC} = 0.9, w_{ICP} = 0.1$ (**0.72**)	same as the best configuration
JHotDraw	2	$w_{CCBC} = 0.7, w_{ICP} = 0.3$ (**0.79**)	$w_{CCBC} = 0.8, w_{ICP} = 0.2$ (**0.77**)
	3	$w_{CCBC} = 0.9, w_{ICP} = 0.1$ (**0.72**)	$w_{CCBC} = 0.8, w_{ICP} = 0.2$ (**0.72**)
	5	$w_{CCBC} = 0.8, w_{ICP} = 0.2$ (**0.69**)	same as the best configuration
SESA	2	$w_{CCBC} = 0.8, w_{ICP} = 0.2$ (**0.93**)	same as the best configuration
	3	$w_{CCBC} = 0.8, w_{ICP} = 0.2$ (**0.70**)	same as the best configuration
	5	$w_{CCBC} = 0.8, w_{ICP} = 0.2$ (**0.66**)	same as the best configuration
SMOS	2	$w_{CCBC} = 0.8, w_{ICP} = 0.2$ (**0.80**)	$w_{CCBC} = 0.9, w_{ICP} = 0.1$ (**0.80**)
	3	$w_{CCBC} = 0.8, w_{ICP} = 0.2$ (**0.77**)	$w_{CCBC} = 0.9, w_{ICP} = 0.1$ (**0.77**)
	5	$w_{CCBC} = 0.9, w_{ICP} = 0.1$ (**0.67**)	same as the best configuration

In parenthesis the reconstruction accuracy, i.e., the average F-Measure

Given these findings we propose the following heuristics to set the parameters of our approach in a real usage scenario:

- *minCoupling*: use the third quartile of the non-zero values of the class-by-class matrix as threshold to remove spurious relationships between the classes of the package to re-modularize.
- *weights*: the weights assigned to the structural and semantic metrics are established for the system under analysis by performing the PCA of the values of the coupling metrics computed on the pair of classes of the system having $ICP > 0$. The value of the proportion of variance obtained for each metric will be used as the weight for the corresponding metric.

5.2 Qualitative Evaluation

Even if our approach is able to reconstruct merged packages with very high accuracy, in a minority of cases it does not reconstruct the original packages and proposes an alternative decomposition of the system. In order to understand if the proposed decomposition is still meaningful, even when different from the original, the developers analyzed the proposed re-modularizations. To select the cases to analyze, we set an F-measure threshold $\epsilon = 0.7$; all the cases under this threshold, i.e., 35 of 150, were analyzed by the developers. Table 6 reports the answers to the question *"Is the proposed package decomposition meaningful?"* given by the developers for each analyzed case. In particular, the table reports the number of students that have answered one of the possible options. Moreover, Table 6 assigns a unique ID to each re-modularization operation evaluated by the students. This is done to easily reference the operations in the discussion of the results. As we can see in Table 6, the developers marked as meaningful most of the re-modularization operations suggested by the tool. It is worth noting that the answers given by the students for each of the analyzed cases never differ by more than one point on the Likert scale, which indicates high agreement among them.

Three of the cases for which the developers gave a positive evaluation will be the object of discussion in the Sections 5.2.3 ($id = 1$), 5.2.4 ($id = 14$), and 5.2.5 ($id = 21$). In this discussion we will use *topic maps* (Kuhn et al. 2007) to represent the main topics in a package. There are many ways to determine topics in source code. In particular, given a generic set of classes S, e.g., a package or a group of packages, it is possible to derive the main topics in S by analyzing the term frequency in the classes it contains. We count, for each term present in S, the number of classes that contain it with a frequency higher than 3. The five most frequent terms, i.e., the terms present in the highest number of classes, are then used to construct the topic map of S 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 S that implements the corresponding topic. The graphical representation of the main topics of S is then obtained by tracing lines between the point on each of the five axes indicating the percentage of classes belonging to S that implement the corresponding topic.

First, we present some of the cases for which the developers gave negative evaluations (i.e., high values on the Likert scale; 4 or 5), in Sections 5.2.1 and 5.2.2.

Table 6 Analysis of the failure cases

System	#Subjects	ID operation	1: Fully agree; 5: Strongly disagree				
			1	2	3	4	5
eTour	2	1	2	–	–	–	–
		2	–	1	1	–	–
		3	–	1	1	–	–
		4	1	1	–	–	–
		5	–	–	2	–	–
		6	–	1	1	–	–
		7	–	–	–	–	2
JHotDraw	2	8	1	1	–	–	–
		9	–	2	–	–	–
		10	–	–	2	–	–
		11	–	1	1	–	–
		12	–	–	2	–	–
		13	–	–	2	–	–
GESA	5	14	4	1	–	–	–
		15	–	–	–	3	2
		16	–	3	2	–	–
		17	–	3	2	–	–
		18	2	3	–	–	–
		19	–	1	4	–	–
		20	–	–	2	3	–
SMOS	5	21	5	–	–	–	–
		22	–	–	–	–	5
		23	–	–	3	2	–
		24	–	–	4	1	–
		25	–	–	3	2	–
		26	–	1	4	–	–
SESA	2	27	–	–	–	–	2
		28	1	1	–	–	–
		29	–	1	1	–	–
		30	–	–	2	–	–
		31	–	–	2	–	–
		32	–	1	1	–	–
		33	–	–	–	2	–
		34	1	1	–	–	–
		35	–	2	–	–	–

Answers to the question: *"Is the proposed package decomposition meaningful?"*

5.2.1 GESA—Merging Two Packages

In a first case negatively evaluated by the developers ($id = 15$ in Table 6) the tool merged the following three packages: *examSessionManagement*, *timetableManagement*, and *classroomManagement*. The last two packages were reconstructed by our approach as a single package. The re-modularization is most likely due to the semantic similarity of the two packages. In fact, the *timetableManagement* package contains, among others, classes for the management of the classroom timetable, which use identifiers similar to those used in *classroomManagement*. Another case where the developers did not agree with the suggested re-modularization is represented by $id = 22$ in Table 6. In this case the tool merged five packages and only four pack-

ages were reconstructed by our approach. In particular, the *connectionManagement* package, which contains the set of classes responsible of the connection to the DBMS, was merged with the *classRegisterManagement* package. This is due to the strong structural relationships (i.e., method calls) between the classes of the two packages. In fact, the *classRegisterManagement* groups all the classes assigned to operations related to the class register, e.g., absence, delay, disciplinary note, and most of these classes access the persistent data in the DBMS using the classes in the *connectionManagement* package.

5.2.2 SESA—Code Clones

Two other interesting cases negatively evaluated by the developers regard the SESA software system ($id = 27$ and $id = 33$ in Table 6). Both these cases have a common reason that caused the failure of our approach. In particular, several classes in SESA contain "source code clones", e.g., pieces of codes copied and pasted among different classes. Moreover, the comments used to describe the responsibilities assigned to the classes follow a standard template containing a set of words shared between almost all the classes of the system. This clearly results in a high semantic similarity even between classes having different responsibilities. Thus, our approach fails to reconstruct the original packages when it attempts to decompose a package created by merging packages containing classes with source code clones and/or very similar comments. A possible solution to this limitation can be found in De Lucia et al. (2011), where the authors propose the use of smoothing filters to improve the performances of the IR-based traceability recovery techniques. In particular, these filters reduce the weight of terms that frequently occur among different artifacts (in our case classes), improving the precision of the IR method. The application of this kind of filters to our approach is outside the scope of this paper. However, we think that our approach could only benefit from the use of the smoothing filters.

5.2.3 eTour—Removing the Preferences Package

A first "failure" case positively evaluated by the developers regards the eTour software system. The following three packages were merged together in a single artificial package:

- *tourOperatorManagement*: this package contains all the classes responsible with the management of the users registered to the system as tour operators.
- *touristManagement*: similar to the previous package, touristManagement groups together all the classes resposible with the management of the users registered as tourists.
- *preferences*: package containing classes used to manage the preferences set by the users of the system, e.g., favored cities for the tourist.

When applied to the resulting package, our technique suggested only two packages, with the six classes from the *preferences* package distributed among the two suggested packages. In particular, two classes were moved to the *tourOperatorManagement* package while four classes were moved to the *touristManagement* package. We

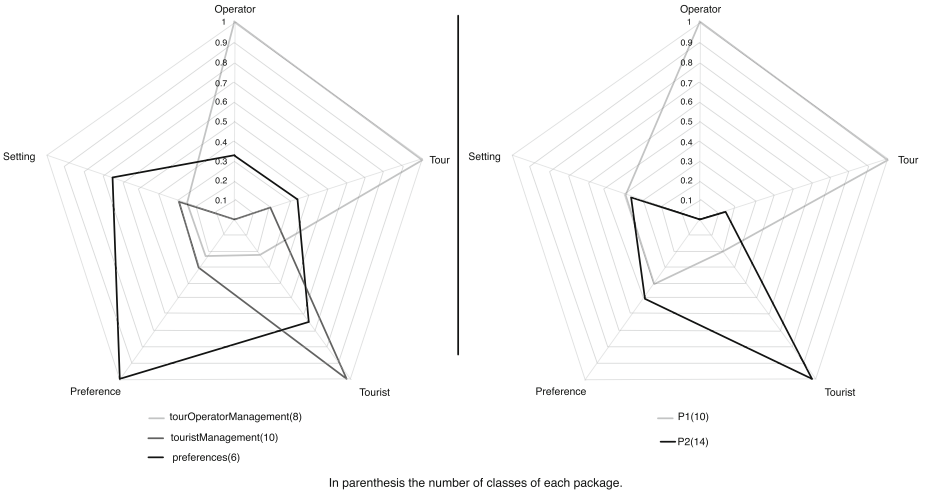

Fig. 4 Topic map eTour: original packages *vs* new packages

deeply analyzed the original package in order to find some explanations. We observed that in eTour three kind of roles are defined for the users: administrator, tour operator, and tourist. However, only the last two have the possibility to customize the system using the preference panel. In fact, two out of the six classes present in *preferences* are responsible for the management of the tour operator's preferences, while the remaining four deal with the tourist's preferences. The high method interactions and semantic consistency between the classes present in the *tourOperatorManagement* (*touristManagement*) package and the two (four) classes responsible of the management of the tour operator's (tourist's) preferences explain the output of our approach. Figure 4 show the topic map of the packages pre and post the re-modularization. It is worth noting that the topics assigned to the new packages are almost the same as the topics assigned to the original *touristManagement* and *tourOperatorManagement* packages.

5.2.4 GESA—Two Packages Instead of Three

In a second interesting case the tool merged in a single package the following three packages, all belonging to the application layer:

- *lessonNegotiationManagement*: GESA has a feature that allows teachers to negotiate the teaching schedule. This package contains all the classes responsible for the schedule negotiation.
- *reportManagement*: this package contains a set of classes that provide different kinds of reports to the administrator of the system. The reports contain a schematic representation of a portion of the persistent data in the system.
- *userManagement*: this package contains all the classes assigned to the management of the system's users.

We applied our approach to reconstruct the original packages. However, the output of the tool was significantly different from the original package decomposition. In fact, instead of three packages the approach reconstructed only two packages; we call them P_1 and P_2. In particular, we noticed that P_1 is equal to the *userManagement* package, i.e., it contains the same set of classes, while P_2 is the result of merging the *lessonNegotiationManagement* package with the *reportManagement* package. To understand if the proposed decomposition is still meaningful we compared the topic map of the original packages with the topic map of the new packages (see Fig. 5). Besides confirming the semantic equivalence of the packages *userManagement* and P_1, from the analysis of Fig. 5 we can observe that in the original decomposition the main topics of *reportManagement* were completely subsumed by the main topics of *lessonNegotiationManagement*. Analyzing the package *reportManagement* we noticed that six out of the seven classes from this package provide the administrator with reports directly or indirectly concerned with the entity lesson, e.g., course timetable report, classroom timetable report, teacher's lesson report, etc. The high conceptual coupling between *lessonNegotiationManagement* and *reportManagement* is also confirmed by the similarity of the topic map representing package *lessonNegotiationManagement* and the topic map representing package P_2 (see Fig. 5). So, while the original decomposition is probably meaningful from a functional point of view, one can argue that the proposed decomposition is still semantically meaningful. This observation is useful in planning our future research, as we could adapt our approach not only to split packages, but also to recommend existing packages to be merged, when needed. As mentioned earlier, design decisions regarding the structure of a system involve more considerations than just high cohesion and low coupling. This is clearly such a case, where the user may or may not agree with our tool's suggestion in the end. It is important to note also that the semantic analysis based on IR techniques, such as, LSI, is independent of grammar and domain models, while it is dependent on consistent use of terms in the analyzed code. Inconsistencies in the use of terms usually affect such an analysis negatively.

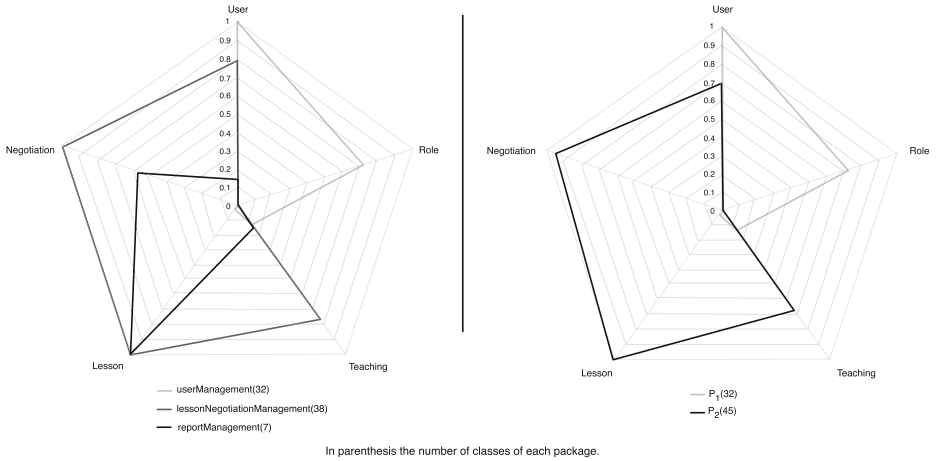

Fig. 5 Topic map GESA: original packages *vs* new packages

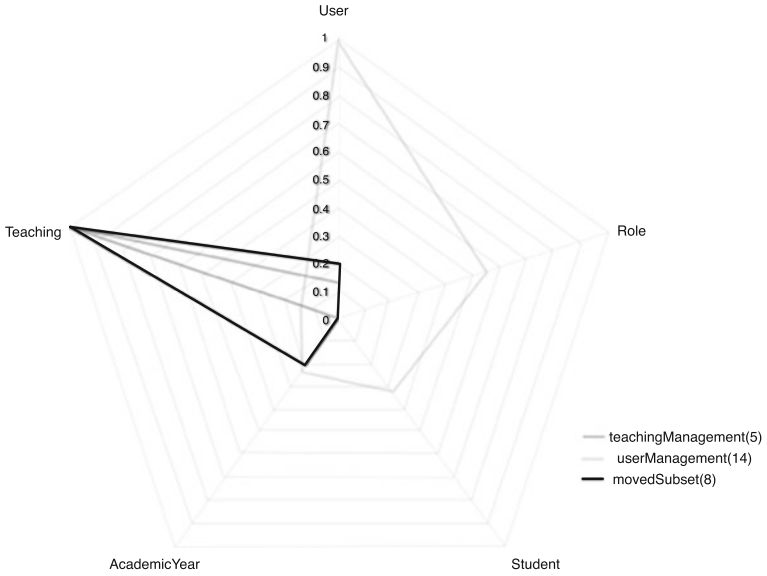

Fig. 6 Topic map of the moved classes in the SMOS re-modularization

5.2.5 SMOS—Moving Classes Between Packages

The last case is from the SMOS software system. In particular, the following packages were merged from the application layer:

- *teachingManagement*: this package contains all the classes assigned to the management of the lectures.
- *userManagement*: this package contains all the classes assigned to the management of the users.

In this case, the number of packages reconstructed is equal to the number of original packages (two), yet our approach applied an interesting operation. Specifically, a subset of eight classes from the *userManagement* package was moved in the *teachingManagement* package. Analyzing this set we noticed that the eight classes implement operations regarding the management of the association between users, i.e. teachers, and teaching assignments (e.g., assign a new teaching assignment to a teacher, show the teacher's assignments, etc.). The topic map shown in Fig. 6 underlines that the moved set of classes is semantically closer to the *teachingManagement* package than to the *userManagement* package. This indicates that probably the set of moved classes is more suited in the teachingManagement package than in the userManagement package. Indeed, although it is different from the original design, the proposed modularization has been evaluated as meaningful by the developers.

6 Threats to Validity

All the findings reported in Section 5 could be affected by several threats to validity (Yin 2003) discussed in the following.

6.1 System Mutation

We decided to use the proposed approach to split previously merged packages and then evaluated the re-modularization accuracy comparing the split packages with the original packages. While the object systems were chosen because they are generally well designed, there is the risk that the original packages are not a good oracle. To mitigate such a threat we analyzed the package decomposition of the subject systems in order to ensure its meaningfulness. Moreover, as we can see in Table 1, the cohesion of the packages in the object systems is very high on average, which is also an indicator of good modularization. In fact, the same metrics were used in Abdeen et al. (2009) to evaluate the decomposition quality of several open source systems, e.g., ArgoUML, JEdit. The metric values obtained by the object systems are much better than the values obtained by the systems in Abdeen et al. (2009). Moreover, JHotDraw is generally considered a well-designed system and it was developed using several design patterns. eTour, GESA, SESA, and SMOS were selected among the best systems developed during a Software Engineering course and, as we can see in Table 1, have an average package cohesion comparable to JHotDraw.

Another aspect of the evaluation is represented by the choice to split previously merged packages. Indeed, given the high quality of the subject systems, the coupling between classes from different packages is generally low. Thus, the splitting operation seems to be trivial. However, we observed that in several cases, classes from different packages have many structural dependencies between them, e.g., such a situation is typical of classes from the subsystems responsible with the management of the system's users. In such cases, the semantic measures avoid the creation of class chains with different responsibilities and help in the reconstruction of the original packages.

6.2 Experiment Design and Results Analysis

The meaningfulness of the proposed re-modularization operations was evaluated using the F-measure,based on the precision and recall that reflect the reconstruction accuracy of the proposed approach. The same approach was also used in previous work on class refactoring (Bavota et al. 2010b, 2010c, 2011b; O'Keeffe and O'Cinneide 2006; Seng et al. 2006).

To better evaluate the suggested re-modularizations, we also analyze the cases where our approach does not reconstruct the original package decomposition. In particular, several students, familiar with the systems, analyzed the proposed alternative re-modularization and evaluated its meaningfulness. The students did not know the goal of our experimentation to avoid bias, however as in any such situation, user subjectivity is part of the evaluation.

6.3 The Role of CCBC in Software Re-Modularization

In order to find the optimal setting of parameters for our approach, we analyzed several different configurations (see Section 4.2). The results showed that the weight w_{CCBC} for the semantic metric should be really high, generally higher than 0.7, to

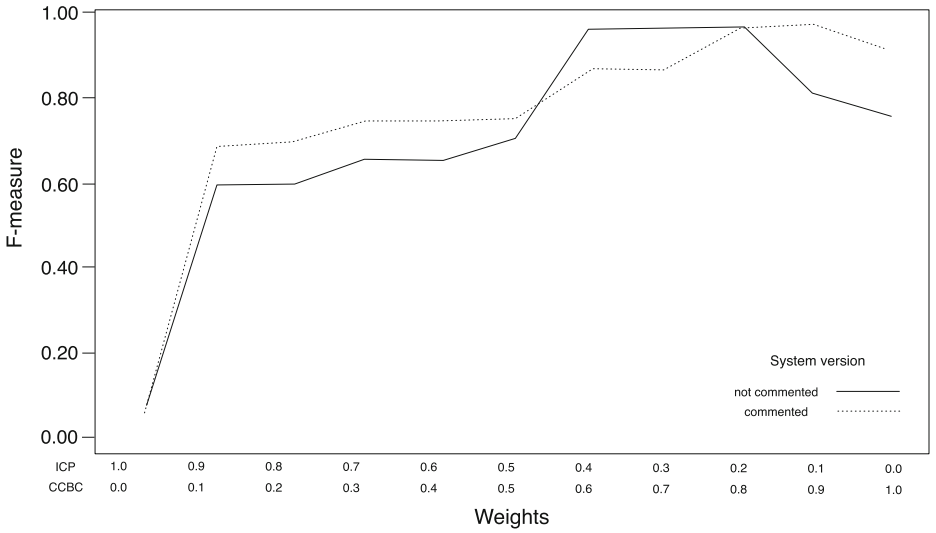

Fig. 7 Performances on $GESA_{\text{Comments}}$ and on $GESA_{\text{NoComments}}$ merging 2 packages

obtain good performances. CCBC is highly dependent on the quality of the identifiers and comments in the code, so, given its high influence on the approach, we expect the approach also to be sensitive to the quality of the comments and identifiers. All the experimented systems have well commented classes besides exhibiting a generally good package decomposition. Thus, to investigate the performances of our approach (and the influence of the configuration parameters) in a completely different scenario, we removed all the comments present in the source code from

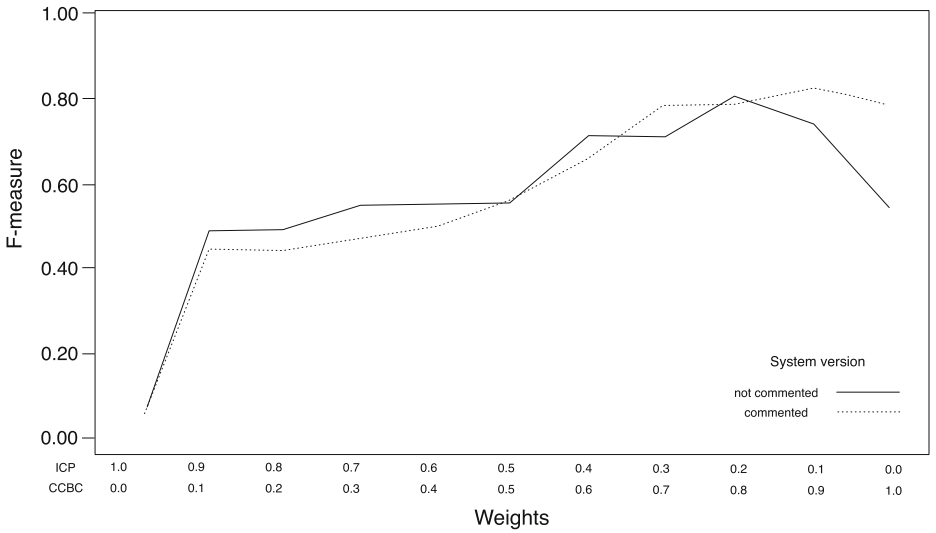

Fig. 8 Performances on $GESA_{\text{Comments}}$ and on $GESA_{\text{NoComments}}$ merging 3 packages

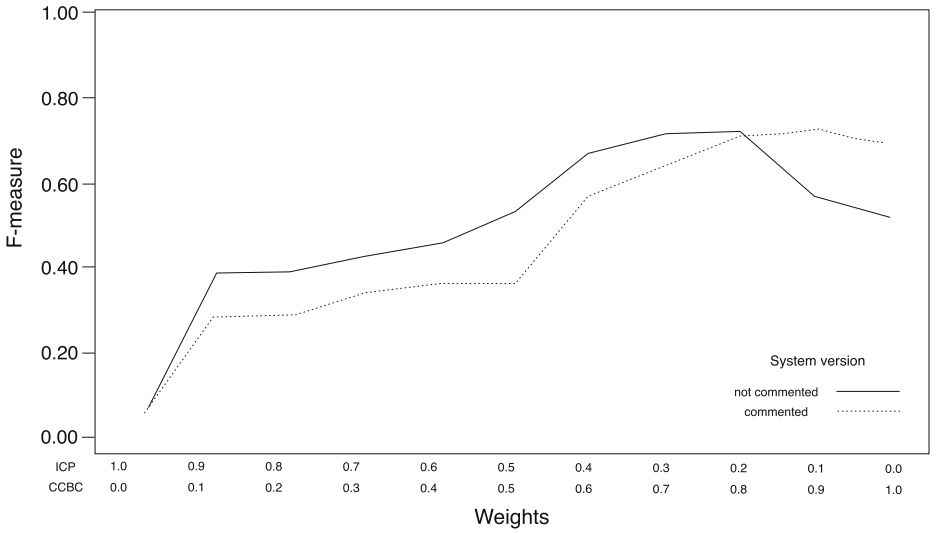

Fig. 9 Performances on $GESA_{\text{Comments}}$ and on $GESA_{\text{NoComments}}$ merging 5 packages

the object systems and re-executed our experimentation. The goal was to assess the impact of the comments on the approach. The results show again that the best performances of our approach are achieved using as threshold the third quartile of the class-by-class matrix, as in the cases where comments are included. Regarding the weights, Figs. 7, 8, and 9 compare the performances of our approach on GESA with comments ($GESA_{\text{Comments}}$) and GESA without comments ($GESA_{\text{NoComments}}$) using all the possible combinations of weights and merging 2, 3, and 5 packages, respectively.[4] It is worth noting that the reconstruction accuracy achieved by our approach is almost identical on the two "versions" of the system. However, our approach applied on $GESA_{\text{NoComments}}$ shows a strong decrease in performance when the weights assigned to the semantic metric is higher than 0.8. Moreover, while the best performances on $GESA_{\text{Comments}}$ are achieved setting $w_{CCBC} = 0.9$, on $GESA_{\text{NoComments}}$ the best results are obtained setting $w_{CCBC} = 0.8$. Thus, even if the weight for the semantic metric slightly decreases, its contribution in the software re-modularization remains essential. This is most likely due to the semantic information present in the identifiers used by the developers. Note that, the need to reduce the weight of the semantic metric in order to achieve good results in the *no comment scenario* is confirmed on all the experimented systems (see Bavota et al. 2011a).

6.4 On the use of PCA as Heuristic to set the Metric Weights

The experimentation performed in the *no comment scenario* allows also to further investigate the validity of the proposed PCA-based heuristic to set the metric weights. In fact, we expected that applying the PCA on the object systems without comments, we obtain a decrement of the proportion of variance assigned to the

[4]The complete results achieved with the other systems can be found in Bavota et al. (2011a)

Table 7 Results of PCA on the "NoComments" systems

	PC1	PC2
(a) eTour		
Proportion of variance	0.55	0.45
Cumulative proportion	0.55	1.00
CCBC	−0.84	0.45
ICP	−0.45	−0.84
(b) GESA		
Proportion of variance	0.80	0.20
Cumulative proportion	0.80	1.00
CCBC	0.99	−0.07
ICP	0.07	0.99
(c) JHotDraw		
Proportion of variance	0.68	0.32
Cumulative proportion	0.68	1.00
CCBC	−0.99	0.07
ICP	−0.07	−0.99
(d) SESA		
Proportion of variance	0.71	0.29
Cumulative proportion	0.71	1.00
CCBC	0.97	0.25
ICP	0.25	−0.97
(e) SMOS		
Proportion of variance	0.84	0.16
Cumulative proportion	0.84	1.00
CCBC	−0.99	0.00
ICP	0.00	0.99

semantic metric, i.e., CCBC, with a consequent decrement of its weight, i.e., w_{CCBC}. To verify such a conjecture, we re-executed the PCA on the "NoComment" versions of the object systems. Table 7 reports the achieved results. Analyzing Figs. 7, 8, and 9, and Table 7, it is easy to see that in this scenario also, the weights suggested by the PCA on the GESA system, i.e., $w_{ICP} = 0.2$ and $w_{CCBC} = 0.8$, result in a configuration with performances very close to the best. This trend is confirmed on all the experimented systems (see Bavota et al. 2011a) and further supports the possibility to use PCA as a heuristic to set the metric weights.

7 Conclusions and Future Work

We proposed and evaluated a technique that suggests decompositions of packages to improve their cohesion. Central to our approach, and a departure from previous work, is the combined use of structural and semantic relationships between classes in this context. The evaluation revealed that the technique produces meaningful decompositions from structural and functional point of view. Due to the heterogeneous nature of software systems, it is always hard to propose analysis techniques that perform equally on any system. Hence, our technique incorporates a PCA-based heuristic that is established empirically for each system on which re-structuring is undertaken. While the heuristic is system dependant, its performance is stable across the systems we experimented with.

Although we applied the approach on five realistic systems, as with all empirical studies, the generalization of our findings cannot be ensured. Thus, replicating the study on other systems is the only way to corroborate the results achieved and mitigate the external threat to validity related to the generalization of our findings. Nevertheless, this paper sheds lights on the effectiveness of combining semantic and structural coupling measures between classes to re-structure packages with low cohesion. At the same time, our work can be used as starting point for future research. We plan to investigate additional structural and semantic measures and determine whether the choice we made here is the best or it can be improved. More empirical studies are planned, using systems with poorer package quality, which would better approximate the everyday use of our technique. A natural extension of this work will be to adapt our technique to be used on multiple, related packages rather than a single one. Coupling between packages (in addition to cohesion) will also have to be considered in such cases. Last but not least, a direct comparison with clustering and search based approaches will be performed.

Acknowledgements We would like to thank all the students who participated to our study. We would also like to thank the anonymous reviewers for their careful reading of our manuscript and useful comments. Andrian Marcus was supported in part by grants from the US National Foundation: CCF-1017263 and CCF-0845706.

References

Abdeen H, Ducasse S, Sahraoui HA, Alloui I (2009) Automatic package coupling and cycle minimization. In: WCRE, pp 103–112

Anquetil N, Lethbridge T (1999) Experiments with clustering as a software remodularization method. In: WCRE, pp 235–255

Antoniol G, Penta MD, Casazza G, Merlo E (2001) A method to re-organize legacy systems via concept analysis. In: IWPC, pp 281–292

Baeza-Yates R, Ribeiro-Neto B (1999) Modern information retrieval. Addison-Wesley, Reading, MA

Basili V, Caldiera G, Rombach DH (1994) The goal question metric paradigm. Wiley, Inc., New York

Bavota G, De Lucia A, Marcus A, Oliveto R (2010a) Software re-modularization based on structural and semantic metrics. In: Proceedings of the 17th working conference on reverse engineering. Beverly, MA, USA, pp 195–204

Bavota G, De Lucia A, Marcus A, Oliveto R (2010b) A two-step technique for extract class refactoring. In: Proceedings of 25th IEEE international conference on automated software engineering, pp 151–154

Bavota G, Oliveto R, De Lucia A, Antoniol G, Guéhéneuc YG (2010c) Playing with refactoring: identifying extract class opportunities through game theory. In: Proceedings of the 26th IEEE international conference on software maintenance

Bavota G, De Lucia A, Marcus A, Oliveto R (2011a) Software re-modularization based on structural and semantic metrics. Tech. rep., University of Salerno. http://www.sesa.dmi.unisa.it/TR2011_EMSE.pdf

Bavota G, De Lucia A, Oliveto R (2011b) Identifying extract class refactoring opportunities using structural and semantic cohesion measures. J syst softw 84(3):397–414

Bittencourt RA, Guerrero DDS (2009) Comparison of graph clustering algorithms for recovering software architecture module views. In: Proceedings of the 2009 European conference on software maintenance and reengineering. IEEE Computer Society, Washington, DC, USA pp 251–254

Canfora G, Cimitile A, De Lucia A, Di Lucca GA (2001) Decomposing legacy systems into objects: an eclectic approach. Inf Softw Technol 43(6):401–412

Cimitile A, Visaggio G (1995) Software salvaging and the call dominance tree. J Syst Softw 28(2):117–127
Corazza A, Martino SD, Scanniello G (2010) A probabilistic based approach towards software system clustering. In: CSMR, pp 88–96
Corazza A, Martino SD, Maggio V, Scanniello G (2011) Investigating the use of lexical information for software system clustering. In: CSMR, pp 35–44
Deerwester S, Dumais ST, Furnas GW, Landauer TK, Harshman R (1990) Indexing by latent semantic analysis. J Am Soc Inf Sci 41(6):391–407
De Lucia A, Oliveto R, Vorraro L (2008) Using structural and semantic metrics to improve class cohesion. In: Proceedings of international conference on software maintenance. Beijing, China, pp 27–36
De Lucia A, Di Penta M, Oliveto R, Panichella A, Panichella S (2011) Improving ir-based traceability recovery using smoothing filters. In: Proceedings of the 19th IEEE international conference on program comprehension. Kingston, ON, Canada, pp 21–30
Ducasse S, Pollet D, Suen M, Abdeen H, Alloui I (2007) Ackage surface blueprints: visually supporting the understanding of package relationships. In: Proceedings of international conference on software maintenance. Paris, France, pp 94–103
Harman M, Hierons RM, Proctor M (2002) A new representation and crossover operator for search-based optimization of software modularization. In: Proceedings of the 2002 conference on genetic and evolutionary computation, pp 1351–1358
Harman M, Swift S, Mahdavi K (2005) An empirical study of the robustness of two module clustering fitness functions. In: Proceedings of the 2005 conference on genetic and evolutionary computation. ACM Press, Washington DC, USA, pp 1029–1036
Hartigan JA (1975) Clustering algorithms. Wiley, New York
Kleinberg JM (1999) Authoritative sources in a hyperlinked environment. J ACM 46(5):604–632
Koschke R, Canfora G, Czeranski J (2006) Revisiting the delta ic approach to component recovery. Sci Comput Program 60(2):171–188
Kuhn A, Ducasse S, Gîrba T (2007) Semantic clustering: identifying topics in source code. Inf Softw Technol 49(3):230–243
Lee Y, Liang B, Wu S, Wang F (1995) Measuring the coupling and cohesion of an object-oriented program based on information flow. In: International conference on software quality
Lehman MM (1980) On understanding laws, evolution, and conservation in the large-program life cycle. J Syst Softw 1:213–221
Maletic JI, Marcus A (2001) Supporting program comprehension using semantic and structural information. In: Proceedings of 23rd international conference on software engineering. IEEE CS Press, Toronto, Ontario, Canada, pp 103–112
Mancoridis S, Mitchell BS, Rorres C, Chen YF, Gansner ER (1998) Using automatic clustering to produce high-level system organizations of source code. In: IWPC, p 45
Maqbool O, Babri HA (2007) Hierarchical clustering for software architecture recovery. IEEE Trans Softw Eng 33(11):759–780
Marcus A, Poshyvanyk D, Ferenc R (2008) Using the conceptual cohesion of classes for fault prediction in object-oriented systems. IEEE Trans Softw Eng 34(2):287–300
Mitchell BS, Mancoridis S (2001) Comparing the decompositions produced by software clustering algorithms using similarity measurements. In: Proceedings of 17th international conference of software maintenance. IEEE CS Press, Florence, Italy, pp 744–753
Mitchell BS, Mancoridis S (2006) On the automatic modularization of software systems using the bunch tool. IEEE Trans Softw Eng 32(3):193–208
O'Keeffe M, O'Cinneide M (2006) Search-based software maintenance. In: Proceedings of 10th European conference on software maintenance and reengineering. IEEE CS Press, Bari, Italy, pp 249–260
Oppenheim AN (1992) Questionnaire design, interviewing and attitude measurement. Pinter Publishers
Poshyvanyk D, Marcus A, Ferenc R, Gyimóthy T (2009) Using information retrieval based coupling measures for impact analysis. Empir Software Eng 14(1):5–32
Praditwong K, Harman M, Yao X (2011) Software module clustering as a multi-objective search problem. IEEE Trans Softw Eng 37(2):264–282
Ricca F, Pianta E, Tonella P, Girardi C (2008) Improving web site understanding with keyword-based clustering. J Softw Maint Evol 20(1):1–29. doi:10.1002/smr.v20:1
Scanniello G, D'Amico A, D'Amico C, D'Amico T (2010) Using the kleinberg algorithm and vector space model for software system clustering. In: ICPC, pp 180–189

Seng O, Bauer M, Biehl M, Pache G (2005) Search-based improvement of subsystem decompositions. In: GECCO, pp 1045–1051
Seng O, Stammel J, Burkhart D (2006) Search-based determination of refactorings for improving the class structure of object-oriented systems. In: Genetic and evolutionary computation conference, pp 1909–1916
Shaw SC, Goldstein M, Munro M, Burd E (2003) Moral dominance relations for program comprehension. IEEE Trans Softw Eng 29(9):851–863
Shtern M, Tzerpos V (2009) Methods for selecting and improving software clustering algorithms. In: Proceedings of 17th IEEE international conference on program comprehension. IEEE CS Press, Vancouver, Canada, pp 248–252
Stevens WP, Myers GJ, Constantine LL (1974) Structured design. IBM Syst J 13(2):115–139
Tonella P (2001) Concept analysis for module restructuring. IEEE Trans Softw Eng 27(4):351–363
van Deursen A, Kuipers T (1999) Identifying objects using cluster and concept analysis. In: Proceedings of 21st international conference on software engineering. ACM Press, Los Angeles, California, USA, pp 246–255
Wiggerts TA (1997) Using clustering algorithms in legacy systems remodularization. In: WCRE '97: proceedings of the fourth working conference on reverse engineering (WCRE '97). IEEE Computer Society, p 33
Wu J, Hassan AE, Holt RC (2005) Comparison of clustering algorithms in the context of software evolution. In: ICSM, pp 525–535
Yin RK (2003) Case study research: design and methods, 3rd edn. SAGE Publications

Gabriele Bavota was born in Napoli (Italy) on November, 19th, 1985. He received (cum laude) the Laurea in Computer Science from the University of Salerno (Italy) in 2009 defending a thesis on Traceability Management advised by Prof. Andrea De Lucia and Dr. Rocco Oliveto. He is currently a PhD student at the Department of Mathematics and Informatics of the University of Salerno under the supervision of Prof. Andrea De Lucia. His research interests include refactoring of software systems, traceability management, information retrieval, software maintenance and empirical software engineering. He serves and has served on in the organizing and program committees of international conferences in the field of software engineering. In particular, he was the web chair of WCRE 2012 and he will be publicity co-chair of ICPC 2013. He is student member of IEEE.

Andrea De Lucia received the laurea degree in computer science from the University of Salerno, Italy, in 1991, the MSc degree in computer science from the University of Durham, UK, in 1996, and the PhD degree in electronic engineering and computer science from the University of Naples "Federico II", Italy, in 1996. He is a full professor of software engineering and the Director of the International Summer School on Software Engineering at the University of Salerno. Previously, he was with the Department of Engineering and the Research Centre on Software Technology (RCOST) at the University of Sannio. He is actively consulting in industry and has been involved in several research and technology transfer projects conducted in cooperation with industrial partners. His research interests include software maintenance, program comprehension, reverse engineering, reengineering, migration, global software engineering, software configuration management, workflow management, document management, empirical software engineering, visual languages, web engineering, and e-learning. He has published more than 150 papers on these topics in international journals, books, and conference proceedings and has edited books and journal special issues. He serves on the editorial board of Journal of Software: Evolution and Process and other international journals and on the organizing and program committees of several international conferences in the field of software engineering. Prof. De Lucia is a senior member of the IEEE and the IEEE Computer Society. He was also at-large member of the executive committee of the IEEE Technical Council on Software Engineering (TCSE) and committee member of the IEEE Real World Engineering Project (RWEP) Program.

Andrian Marcus is Associate Professor and Director of the Undergraduate Program in the Department of Computer Science at Wayne State University (Detroit, MI). He obtained his Ph.D. in Computer Science from Kent State University in 2003. His current research interests are in software engineering, with focus on using information retrieval and text mining techniques for software analysis to support comprehension during software evolution. He served on the Steering Committee of the IEEE International Conference on Software Maintenance (ICSM) between 2005–2008 and

2011–2014, and on the Steering Committee IEEE International Wokshop on Visualizing Software for Understanding and Analysis (VISSOFT) between 2005–2009. He serves on the editorial board of the Empirical Software Engineering and the Journal of Software: Evolution and Process. He also served as organizing or program committee member to many conferences related to his area of research.

Rocco Oliveto received (cum laude) the Laurea in Computer Science from the University of Salerno (Italy) in 2004. From October 2006 to February 2007 he has been a visiting student at the University College London, UK. He received the PhD in Computer Science from the University of Salerno (Italy) in 2008. From July 2009 to September 2009 he has been a visiting researcher at the Polytechnique of Montreal, Canada. From July 2009 to September 2009 he has been a visiting researcher at the Polytechnique of Montreal, Canada. From 2008 to 2010 he was research fellow at the Department of Mathematics and Informatics of the University of Salerno. From 2005 to 2010 he is also adjunct professor at the Faculty of Science of the University of Molise (Italy). He is member of the Software Engineering Lab at the University of Salerno. In 2011 he joined the STAT Department of the University of Molise where he is currently assistant professor. His research interests include traceability management, information retrieval, software maintenance and evolution, and empirical software engineering. He has published about 40 papers on these topics in international journals, books, and conference proceedings. He serves on the editorial board of the Advances in Software Engineering. He serves and has served as organizing and program committee member of international conferences in the field of software engineering. In particular, he was the program co-chair of TEFSE 2009, the Traceability Challenge Chair of TEFSE 2011, the Industrial Track Chair of WCRE 2011, the Tool Demo Co-chair of ICSM 2011 and he will be the program co-chair of WCRE 2012. Dr. Oliveto is member of IEEE, ACM, and IEEE-CS Awards and Recognition Committee.