Remodularization Analysis using Semantic Clustering

Gustavo Santos
and Marco Tulio Valente
Department of Computer Science
UFMG, Brazil
{gustavojss, mtov}@dcc.ufmg.br

Nicolas Anquetil
RMoD Team
INRIA, Lille, France
nicolas.anquetil@inria.fr

***Abstract*—In this paper, we report an experience on using and adapting Semantic Clustering to evaluate software remodularizations. Semantic Clustering is an approach that relies on information retrieval and clustering techniques to extract sets of similar classes in a system, according to their vocabularies. We adapted Semantic Clustering to support remodularization analysis. We evaluate our adaptation using six real-world remodularizations of four software systems. We report that Semantic Clustering and conceptual metrics can be used to express and explain the intention of the architects when performing common modularization operators, such as module decomposition.**

***Index Terms*—Software Architecture; Software Maintenance; Remodularization; Text Processing; Information Retrieval.**

I. Introduction

As a software evolves, its structure inevitably gets more complex and harder to understand and modifications become more difficult to implement. Remodularization tasks are then recommended to reduce the effort required in future maintenance and to restore the original architecture. More specifically, we define as remodularization a major restructuring in the system's architecture, with the central goal of improving its internal quality and therefore without adding new features or fixing bugs. In most cases, remodularizations are guided by structural aspects, like static dependencies between architectural entities [1, 2]. For example, a common recommendation is that cohesion should be maximized and coupling should be minimized, which can be followed manually or with the help of semi-automatic remodularization tools [3].

Despite that general recommendation, the fact is that there is no solid agreement on what constitutes a good architecture. Architectural quality is a subjective concept that is difficult to measure by conventional quality metrics. For example, recent work started to question the structural cohesion/coupling dogma, stating that "coupling and cohesion do not seem to be the dominant driving forces when it comes to modularization" [2, 4]. Other work showed that structural cohesion metrics usually present divergent results when used to evaluate the same refactoring actions [5]. On the other hand, there is also recent research proposing a new set of metrics for both cohesion and coupling, named conceptual quality metrics. Such metrics are calculated considering lexical information as described in source code identifiers and comments [6, 7, 8]. In fact, by just reading the identifiers and comments in a block of code even an unfamiliar software maintainer can usually get an initial understanding of its semantics and intention [9]. The terms extracted from identifiers and comments are usually referred as the system's vocabulary.

In this paper, our goal is to investigate an approach for evaluating architectural quality using conceptual metrics. More specifically, we report an experiment on using and adapting Semantic Clustering to evaluate software remodularizations. Semantic Clustering is an approach that relies on information retrieval and clustering techniques to extract sets of similar classes in a system according to their vocabularies [9, 10]. We adapted Semantic Clustering to extract different numbers of clusters, depending on the variability in the vocabulary. We also adapted the technique to support the comparison of two versions of a system before and after a given remodularization effort, regarding their vocabularies.

We evaluate our adaptation of Semantic Clustering using six real-world remodularizations of four different systems. We extracted clusters for each remodularization and evaluated them using a set of metrics that measure conceptual cohesion, spread, and focus [11]. Recent work presented modularization operators that are likely to occur in any remodularization. We also map major changes in conceptual quality with these modularization operators. As already reported in other studies [2, 12], we show that module decomposition is a common operation in remodularizations. In this case, the new packages are usually more cohesive than the original package (in conceptual terms) and they also exhibit a higher degree of focus (i.e., usually there is a major and dominant concept in the new packages). On the other hand, the values of spread usually increase, because there are more packages associated to the concept.

The contributions of this paper are:

- We propose a methodology to evaluate software remodularizations using Semantic Clustering. We address the configuration of the approach in order to obtain better clusters and we also present a tool that supports our adapted methodology.
- We analyze the occurrence of modularization operators in four systems, by correlating major changes in conceptual quality to the operators that were applied. We found

that module decompositions are the operations with the highest impact on semantic clusters. This operation tends to increase both the spread and focus of the semantic clusters and to derive new packages with higher conceptual cohesion (> 0.40).

The remainder of this paper is organized as follows. Section II presents a catalog of common modularization operators in software maintenance. Section III describes the original Semantic Clustering technique. Section V describes an application of this technique to remodularization analysis. Section VI describes the conceptual metrics we used in our analysis. Section VII presents the study on the remodularizations of JHotDraw and Section VIII discuss threats to validity. Finally, Section IX presents related work and Section X concludes.

II. Remodularization Definition and Operators

Remodularization, or Large Scale Refactoring in Fowler's definition, is a major change in design and implementation restrict to architecture [13]. It is performed in order to organize the architectural entities in modules, with well-defined interfaces, and preserving the code's behavior. The new organization might consider different aspects of relationship: common change, high coupling or functional aggregation [14, 15].

Although the importance of remodularization is quite well-known, it is a time and personnel demanding task. It requires a lot of program comprehension in order to point the changes to be made. Unlike small scale refactoring, there are no fast refactoring examples or some variety of tools to support remodularizations [12]. Indeed, remodularization is often applied in an advanced stage of architectural erosion [2, 16].

Rama and Patel formalized six elementary operators most likely to occur in any remodularization [12]. They validated these operators in three remodularization cases, discussing specific situations to which every operator was applied. A short description of each operator is given as follows:

- **Module Decomposition (MD):** The most common operator basically consists in separating a module into new smaller modules.
- **Module Union (MU):** The opposite operator of Module Decomposition consists in creating a bigger module from smaller ones.
- **File Transferal (FT):** Typically, a file is transfered to another module, which is similar to Move Class [13].
- **Function Transferal (FuT):** Similar to Move Method, this operator moves a function from one file to another [13].
- **Promote Function to API (PF):** In OO systems, this operator increases the visibility of a method (from private to public, for example).
- **Data Structure Transferal (DT):** Basically, this operator moves one attribute to another file.

Our study maps major conceptual changes in the architecture using these operators. We focus on architectural changes, thus it is expected that the first three operators (Module Decomposition, Module Union, and File Transferal) must be the most occurring operators in our evaluation. In fact, when creating modules or moving classes from modules, we are actually re-defining the module's vocabulary. Moreover, we consider one more operator, restrict to conceptual analysis and defined in Fowler's refactoring catalog [13]:

- **Rename (RN):** Changes the name of an entity, when its former name is not intuitive for developers.

Although we looked for these modularization operators in four systems, we do not propose a methodology to identify refactoring opportunities. We focus on how these operators actually impact the conceptual quality after a remodularization. This study is presented in Section VII.

III. Semantic Clustering

Semantic Clustering is an Architecture Recovery technique originally proposed by Kuhn *et al.* [9, 10] to extract sets of similar classes in a system, according to the lexical similarity of their vocabularies.[1] It also supports a visualization on how these sets are scattered in the system's package structure. Four main functions are proposed to generate semantic clusters: text *extraction and weighting* from source code, term *indexing* with Latent Semantic Indexing, *clustering* classes with similar vocabularies, and *visualization* of how the proposed clusters are distributed over the package structure. In the following paragraphs, we detail these functions.

Extraction and Weighting: Semantic Clustering views classes as documents, and the vocabulary of a class is extracted from identifiers and comments. Thus, keywords from the programming language are discarded. Terms are obtained by splitting identifiers with the camel case and underscore naming convention. For example, *ClassName* is separated into *class* and *name*. Common words that do not add meaning to the text, called *stopwords*, are excluded. A stemming function is then applied to remove affixes and suffixes. For example, *generation*, *generate*, and *generated* have the same stem *generat*. As a result of this preprocessing task, a term-document matrix is created. Each row of this matrix represents a distinct term in the vocabulary, and each column represents a class. Terms are weighted using a *tf-idf* function to punish words that appear in many documents [17].

Indexing: Given a term-document matrix, Semantic Clustering relies on Latent Semantic Indexing (LSI), an information retrieval technique widely used in many applications, such as feature location and program comprehension [18, 19]. LSI is used to represent the term-document matrix in a smaller number of terms and with minimal loss of information [20, 21].

Clustering: The clustering function works on this compacted matrix, in which each document is represented as a vector

[1] In this paper, we decide to use the original name of the technique, as proposed by Kuhn *et al.* However, we acknowledge that the term *semantic* in this case denotes lexical-based relations.

and the similarity of a pair of documents is calculated by the cosine of the smallest angle formed by such vectors. After calculating the similarity between each pair of documents, an agglomerative hierarchical clustering algorithm is executed. Each class is assigned to a single cluster that represents a domain concept. The technique also generates a small set of relevant terms, called semantic topics, which represents the meaning (or intention) of each cluster.

Visualization: The clusters resulted of Semantic Clustering are visualized through the use of Distribution Maps [11]. A Distribution Map is a visualization in which packages are displayed as boxes, with filled squares representing the package's classes. The color of a class represents the cluster to which the class belongs. In a Distribution Map, two kinds of information are displayed: (i) the *structural* information denotes how the classes are organized in packages; (ii) the *conceptual* information relates to the distribution of semantic clusters through classes' colors.

IV. Semantic Clustering Stop Criterion

Kuhn *et al.* presented a case study in which they propose a fixed number of nine clusters for Semantic Clustering [9]. However, in our initial experiments with the technique, we observed that the number of clusters depends on the size of the system. Otherwise, for large systems with hundreds of classes, the generated clusters will contain many classes and therefore they will be difficult to analyze. To address this issue, we propose to stop the hierarchical clustering using a similarity threshold.

The proposed agglomerative hierarchical clustering approach starts with each class in an unitary cluster. At each step of the algorithm, the pair of clusters with the highest similarity is merged. After that, all similarities are recalculated, until a given number of clusters is obtained. In this case, we replaced the stop criteria to be based on a threshold: pairs of clusters will be merged until all pairs have similarity lower than a given threshold. Thus, the algorithm stops with the current configuration of clusters.

This strategy ensures that different number of clusters can be extracted from different systems, depending on how the systems' vocabularies are similar to each other. Despite that, we still have to define this threshold. Particularly, a high threshold should not be used in systems whose classes are loosely similar, since the algorithm will stop early with a large number of clusters. To address this issue, we propose to execute the clustering algorithm several times, changing the threshold value. Finally, we select the threshold that produces highly cohesive clusters (according to CCCluster metric, described in Section VI) and in a plausible number.

V. Semantic Clustering for Remodularization Analysis

In this section, we present our application of Semantic Clustering in order to support remodularization analysis. First, we present in Section V-A the algorithm to compare two versions of a given system with a conceptual point of view. Next, Section V-B presents the use of Distribution Maps to support visual analysis. And finally, we report a prototype tool that supports our improvements to Semantic Clustering in Section V-C.

A. Semantic Clusters Comparison

As described in Section III, we visualize the clusters that Semantic Clustering generates for a system in a Distribution Map. However, in this work we want to compare the semantic clusters before and after a remodularization. For this purpose, a naive solution would be to apply Semantic Clustering separately for each version and to compare the generated clusters. However, these clusters might not be comparable. The creation and deletion of classes might change the vocabulary and create new similarities, which can lead to new clusters. Thus, the number of clusters before and after might not be the same.

We propose an algorithm to support the comparison of clusters computed by Semantic Clustering, regarding different versions of a system. By using this algorithm, Semantic Clustering is applied only to the first version (before the remodularization), and then we map every class in the newer version to a previously calculated cluster. The algorithm is presented in Figure 1. Basically, it receives as input the clusters extracted for the version before the modularization ($Clusters_{before}$) and the classes in the version after the modularization effort. The algorithm generates as output a new list of clusters ($Clusters_{after}$). In this list, each class in the newer version is mapped to one of the original clusters.

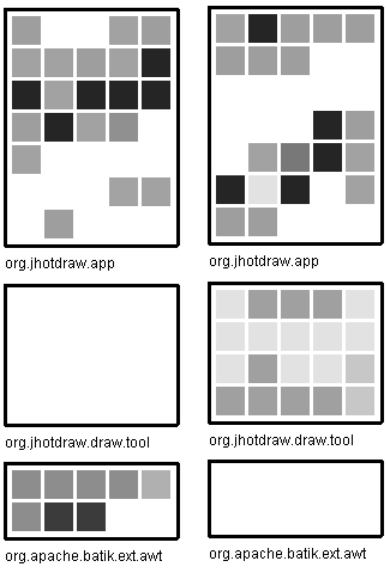
Input: $Clusters_{before}, C_{after}$
Output: $Clusters_{after}$
1: $Clusters_{after} = \emptyset$
2: **for** $c \in C_{after}$ **do**
3: $bestSimilarity = -\infty$
4: $bestCluster = -1$
5: **for** $cluster \in Clusters_{before}$ **do**
6: $\vec{v}_{class} = classAsVector(c)$
7: $\vec{v}_{cluster} = clusterAsVector(cluster)$
8: $s = cosineSimilarity(\vec{v}_{cluster}, \vec{v}_{class})$
9: **if** $s > bestSimilarity$ **then**
10: $bestSimilarity = s$
11: $bestCluster = cluster.index$
12: **end if**
13: **end for**
14: $assign(Clusters_{after}(bestCluster), c)$
15: **end for**

Fig. 1. Distribution Map Comparison Algorithm

As discussed in Section III, a column of the term-document matrix represents a class. Thus, for each class of the newer version of the system, we extract its representing column of the matrix (line 6). A cluster is not represented in the matrix, so we compute its vector as the sum of the vectors of its classes (line 7). Then, the algorithm tests the similarity of a class and the cluster candidates. This similarity is calculated as the cosine similarity of their respective vectors (line 8). Each class is then assigned to the cluster that returns the best similarity in the computed tests (line 14).

B. Visualization Support

The result of the comparison algorithm is used to generate two Distribution Maps, as presented in Figure 2. In these maps, the location of a class in a package is fixed. Classes added after the remodularization are displayed as blank squares in the first Distribution Map. Similarly, classes removed after the remodularization are displayed as blank squares in the second Distribution Map. For example, by analyzing the Distribution Maps, it is possible to visualize that eleven classes were created in `org.jhotdraw.app` package, and twelve classes were removed. As another example, the `org.jhotdraw.tool` package was created with twenty classes. Similarly, the package `org.apache.batik.ext.awt` was removed.



(a) Before (b) After

Fig. 2. Distribution Maps before and after remodularization in JHotDraw

C. Tool Support

We implemented a prototype tool, called TopicViewer[2], that supports our improvements to Semantic Clustering (as described in Section V-A). TopicViewer's architecture includes components for text extraction from source code, Semantic Clustering operations and Distribution Map displaying. The tool is a desktop application and was designed to allow its use with other textual documents, like bug reports, external documentation, task definitions, and commits.

VI. Conceptual Metrics

This section presents the conceptual metrics we used in our case studies. In this work, we do not make use of coupling metrics. As stated by Taube-Schock et al. [22], high coupling is not avoidable, and even natural, during the evolution of a software. In addition to the cohesion metrics, we also used two metrics for measuring the concentration and scattering of semantic clusters across the package structure. Both metrics were proposed by Ducasse et al. and can be applied to Distribution Maps.

- **Conceptual Cohesion of a Cluster (CCCluster):** This metric is a straightforward extension of the Conceptual Cohesion of a Class (C3) metric, proposed by Marcus and Poshyvanyk [6]. C3 is calculated as the average cosine similarity of each pair of methods in a given class. Similarly, CCCluster is the average cosine similarity of each pair of classes in a *cluster*. Moreover, the internal cohesion of a clustering is the average CCCluster of all generated clusters.
- **Conceptual Cohesion of a Package (CCP):** This metric is similar to CCCluster metric, but it is the average cosine similarity of each pair of classes in a given *package*. Also, note that CCP and CCCluster measure cohesion of different groups of classes: packages and clusters, respectively. The classes of a package can be covered by different clusters, as well as one cluster can cover classes of different packages. We apply this metric to evaluate the cohesion of packages between two versions of a system, before and after a remodularization.
- **Spread:** Basically, it computes the number of packages in which at least one class is covered by a given cluster. We expect a decrease in this metric after a remodularization. This is related to change impact, in the sense that a concept should be less scattered to reduce future maintenance work.
- **Focus:** Measures the distance between the distribution of a cluster and the package structure. Given a cluster, if its focus is close to one, then it covers the majority of the classes of the packages it touches. We expect an increase in this metric. The rationale is also related to change analysis: if we have a concept that is concentrated in few packages, then it will be easier to maintain.

VII. Case Study

The *goal* of our study is to verify whether conceptual aspects, as expressed by the clusters retrieved by Semantic Clustering, actually reflect an increasing in quality after real remodularizations. Thereby, we focus on identifying major changes in conceptual quality properties and explain these changes under the *perspective* of typical remodularization operations. More specifically, we intend to provide answers and insights for the following research questions:

- RQ #1: *What is the impact of remodularizations in the clusters generated by Semantic Clustering?* Basically, we aim to compare for example the clusters before and after remodularizations, using focus and spread.
- RQ #2: *What are the remodularization operations that have more impact in the clusters generated by Semantic Clustering?* We intend to classify the refactoring operations responsible by the major impacts in spread and focus, detected when answering the first question.

[2]http://code.google.com/p/topic-viewer.

- RQ #3: *What is the impact of recurrent modularization operators in terms of conceptual cohesion?* With this research question, we intend to establish correlations between recurrent remodularization operators—specially those with a relevant impact in the clusters generated by Semantic Clustering—and conceptual cohesion metrics.

We describe our study as follows. Section VII-A presents the remodularization dataset and Section VII-B describes the preparation and cleaning of this dataset. After that, Sections VII-C to VII-D present the study results.

A. Target Systems

Our evaluation relies on six remodularization cases, concerning four Java-based systems:

- **Eclipse** went through a substantial remodularization to integrate the OSGi technology. Existing features of Eclipse were separated into new components during two remodularizations, from versions 2.0.1 to 2.1, and from versions 2.1 to 3.0. The Eclipse case consists in a major and global remodularization.
- **JHotDraw** also had two remodularizations, from versions 7.3.1 to 7.4.1, and from versions 7.4.1 to 7.5.1. The first one is global, regarding the number of created packages; and the last one is local, i.e., it impacted a few number of packages.
- **NextFramework** is a web-based development framework.[3] The system's remodularization is very similar to Eclipse, since it also happened to move to the OSGi technology. The remodularization was globally applied to the system's architecture.
- **Vivo** is an open-source researcher networking and collaborative platform.[4] The remodularization we considered was restricted to the subsystem Vitro, from version 1.4.1 to 1.5. This remodularization is similar to JHotDraw's: a local restructuring changing a small number of packages.

Table I provides descriptive statistics of these systems and their versions. For each one, we calculated the vocabulary considering identifiers, i.e., class, attributes, and method names, and also comments and documentation ($|V_{all}|$). Moreover, we calculated the average number of terms per class ($|V_{all}|$ / NOC). An average number of terms per class around two increases the performance of LSI, since this technique relies on high dimensional data. Based on our experience, the lower the number of terms per class, the more likely we obtain larger clusters and many small (even unitary) clusters.

B. Methodology

This section details the steps we followed in the evaluation of each system of our dataset.

[3]http://www.nextframework.org.
[4]http://vivoweb.org.

TABLE I
VOCABULARY DATA (NOC= # OF CLASSES; NOP= # OF PACKAGES)

System	KLOC	NOP	NOC	V_{all}	V_{all}/NOC
Eclipse 2.0.1	420	104	2,331	3,414	1.46
Eclipse 2.1	494	110	2,620	3,771	1.44
Eclipse 3.0	599	142	3,138	3,741	1.19
JHotDraw 7.3.1	126	46	715	1,878	2.63
JHotDraw 7.4.1	125	62	715	1,807	2.53
JHotDraw 7.5.1	134	64	748	1,856	2.48
Next 12-08-07	56	52	536	1,449	2.70
Next 12-08-22	67	73	607	1,487	2.45
Vivo 1.4.1	142	91	899	1,902	2.12
Vivo 1.5	147	95	940	1,920	2.04

Isolating the remodularization: In order to attend the definition of remodularization proposed in Section II, we first isolated the remodularization work from enhancements or functional additions. The rationale is that new features typically introduce new terms to the vocabulary, which may result in the creation of new clusters. To tackle this issue, we manually examined the packages created after the remodularization, including their documentation. Packages that introduce new features or packages created specifically to support a new technology (e.g., a package whose classes provide integration to OSGi technology, for example) were discarded from our analysis. Test classes were discarded as well. The data presented in Table I refers to the systems after discarding the aforementioned packages.

Semantic clustering: After preparing the dataset, we executed the semantic clustering algorithm several times for each version before the considered remodularizations, changing only the similarity thresholds in increments of 0.05. Table II presents the thresholds that generated the highest cluster cohesion and a reasonable number of clusters, as discussed in Section IV. For example in Eclipse-3.0 a similarity threshold of 0.65 resulted in the best cluster cohesion, but also in 247 clusters. In this case, we chose the second best threshold, 0.60, which resulted in 175 clusters. Finally, to compare the versions before and after the considered remodularizations, we relied on the algorithm described in Section V-A.

TABLE II
THRESHOLDS (THR) SELECTION (CLU= # CLUSTERS)

System	Thr	Clu	CCCluster	NOC / Clu
Eclipse 2.0.1	0.60	132	0.71	18
Eclipse 2.1	0.55	109	0.68	25
Eclipse 3.0	0.60	175	0.68	18
JHotDraw 7.3.1	0.60	44	0.79	17
JHotDraw 7.4.1	0.60	43	0.80	17
JHotDraw 7.5.1	0.65	57	0.78	14
Next 12-08-07	0.60	27	0.80	20
Next 12-12-11	0.60	22	0.83	28
Vivo 1.4.1	0.55	34	0.75	27
Vivo 1.5	0.65	75	0.76	13

Computing conceptual metrics: For each remodularization, we

calculate the conceptual metrics presented in Section VI, including a metric applied to packages (Conceptual Cohesion of a Package); and two metrics measuring properties of semantic clusters (Spread and Focus).

C. Impact of Remodularizations on Semantic Clusters (RQ #1)

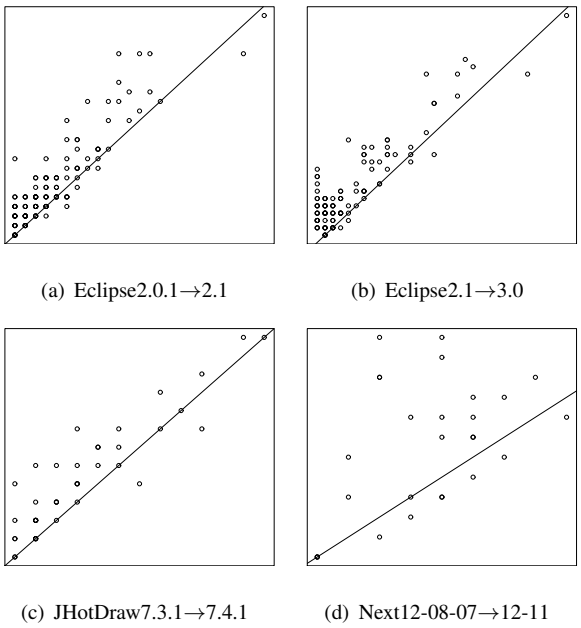
To foster comparison, our approach generates the same semantic clusters for the versions before and after a remodularization (although the classes in the clusters may change). For each cluster, we then calculate its spread and focus, considering the versions before and after the remodularization, as presented in Figures 3 and 4. We restricted this analysis to global remodularizations (see Section VII-B). Each point in the scatterplots represent a concept, as extracted by Semantic Clustering. The horizontal axis shows the metric values before the remodularization, and the vertical axis shows the values after the remodularization work. A point above the diagonal means that the metric value for its representing concept increased after the remodularization. Similarly, a point below the diagonal line means that the metric value decreased.

For most clusters, the spread values increased after the remodularization, as most of the points are above the diagonal line. In fact, since the concepts are the same before and after a remodularization and considering that new packages were created (as can be figured out in Table I), the existing concepts will appear in more packages after the remodularization. Concerning focus, we can observe in Figure 4 that the distributions are not clear, since we have a considerable number of clusters both above and below the diagonal line in all presented remodularizations.

We also applied a nonparametric Wilcoxon test to compare the remodularizations under analysis, according to the spread and focus of the extracted clusters. Thereby, after the Wilcoxon test, if the p-value is lower than 0.05, then the clusters are statistically nonidentical and we can claim that the remodularization actually changed the conceptual structure of the systems, in terms of focus or spread.

Table III presents the Wilcoxon test results (values in bold are statistically significant). The results show that the increasing of spread is statistically significant in the entire dataset. On the other hand, confirming our previous observation, we can not assure the same regarding focus. Although we have some increasing in most remodularizations, the Wilcoxon results only provide statistical support to the last remodularization of Eclipse and the first remodularization of JHotDraw. Most clusters of the first remodularization of JHotDraw increased their focus.

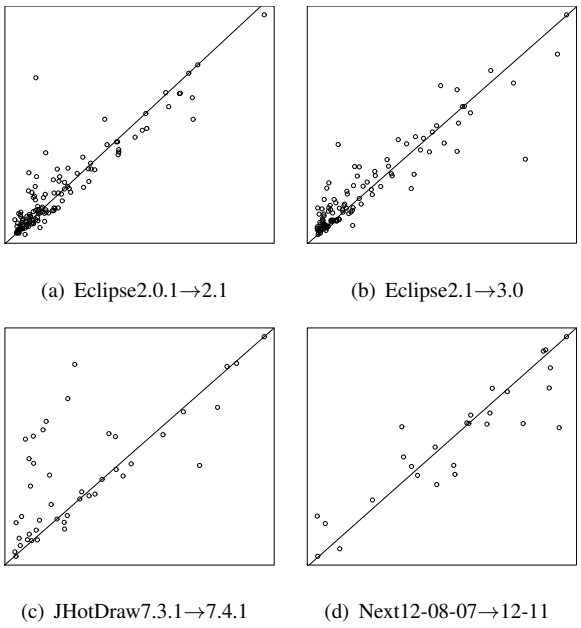
Summary: Remodularizations tend to consistently increase the spread of the existing concepts among the new package structure. Regarding focus, there is no clear tendency, and it is possible to have clusters both with an increase or with a decrease in focus. Therefore, focus is not an appropriate quality metric in this setting.



(a) Eclipse2.0.1→2.1 (b) Eclipse2.1→3.0

(c) JHotDraw7.3.1→7.4.1 (d) Next12-08-07→12-11

Fig. 3. Spread results (the spread of the concepts *above* the diagonal increased after the remodularization)



(a) Eclipse2.0.1→2.1 (b) Eclipse2.1→3.0

(c) JHotDraw7.3.1→7.4.1 (d) Next12-08-07→12-11

Fig. 4. Focus results (the focus of the concepts *above* the diagonal increased after the remodularization)

D. Remodularization Operators with Highest Impact in Semantic Clusters (RQ #2)

To address this research question, we focused on the modularization operators proposed by Rama and Patel [12] (see Section II). For each remodularization, we collected the three clusters with the highest increase and decrease in spread and focus, separately. We carefully analyzed the source code of the classes in each selected cluster, before and after the remodu-

TABLE III
WILCOXON TEST RESULTS FOR SPREAD AND FOCUS OF CLUSTERS (MEAN INCR.= MEAN INCREASE)

Metric	Remodularization	Mean Incr.	p-value
Spread	Eclipse 2.0.1→2.1	1.61	**<0.001**
	Eclipse 2.1→3.0	2.82	**<0.001**
	JHotDraw 7.3.1→7.4.1	0.98	**<0.001**
	Next-12-08-07→12-12-11	2.04	**0.001**
Focus	Eclipse 2.0.1→2.1	0.01	0.408
	Eclipse 2.1→3.0	0.02	**<0.001**
	JHotDraw 7.3.1→7.4.1	0.09	**0.008**
	Next 12-08-07→12-12-11	-0.02	0.518

larizations, aiming to identify the operations that explain the change in focus or spread. To reduce the amount of manual analysis, we decided to inspect one remodularization of each system. For this reason, we removed two remodularizations: (a) Eclipse 2.0.1→2.1, because it presented the highest p-value regarding the global results, as showed in Table III; (b) JHotDraw 7.4.1→7.5.1, because it consisted in just local and minor remodularizations.

Table IV summarizes the results of our analysis of 47 clusters (2 metrics x 6 clusters x 4 systems). We only found two clusters with bottom results for spread in JHotDraw, i.e., the spread of the other clusters remained constant or increased after the remodularization. For each cluster, we present a brief description of the concepts behind the cluster and the remodularization operator responsible for the change under analysis, when it was possible to identify such operator. For example, we were not able to explain the bottom results for focus in JHotDraw in terms of modularization operators.

We applied Pearson chi-squared test to analyze the association of (a) the occurrence of a modularization operator and (b) an increase or decrease in one metric, considering the 47 cases in Table IV. After the test, if the p-value is lower than 0.05, these two descriptive variables a and b are dependent. We only applied this test for Module Decomposition, because it was the most frequent operator in the experiment.

Our main findings when investigating this research question are described as follows:

- Module Decomposition (MD) was the operator responsible for most distinguished changes in spread and focus, covering 24 out 47 clusters we selected for analysis. The operator was responsible for an increase in spread, in 9 out 12 clusters. Regarding focus, the operator explains the observed increments in focus in 10 out of 12 clusters we manually inspected. The chi-squared test showed that the occurrence of module decomposition (a) has statistic association with the Top-3 increasing of spread (b_1) and focus (b_2), with p-values 0.022 and 0.001, respectively.
- The transferal of files (FT) and data structures (DT) was identified in five cases. In two of them, there was a decrease in spread. This fact confirms the motivation of these operators: moving one entity that is misplaced in the architecture to a more similar module.
- Rename (RN) was performed in Next and Vivo to correct typos, e.g., *Sumary* and *PropStmt*, to *Summary* and *PropertyStatement*, respectively. There was a decrease in focus in three out of four rename operations, because new similarities were built with other classes in which the terms are correctly spelled.
- Module Union (MU) occurred along with an increase (with Module Decomposition) and decrease in Focus (with Rename). Promote Function (PF) was applied only once, with an increase in focus also with Module Decomposition.
- We also identified other operations: file (FR) and module (MR) removal in four cases. They were necessary to provide a better interface for color gradients in JHotDraw and code loaders in Next. In these specific cases, there was a decrease in spread.

Summary: Module decomposition is commonly the operator behind the increasing in spread and focus. For other operators, we can not draw statistic relationship between the operator and an improvement or decline of a metric.

E. Impacts of Package Decomposition in Conceptual Cohesion (RQ #3)

With this question, we aim to investigate whether the cohesion of the packages directly impacted by remodularizations is greater than the cohesion of the original packages. More specifically, we concentrate the analysis in package decomposition operations, since it was the operation responsible by the major changes in semantic clusters, after remodularizations. In this case, an original package P is decomposed in new packages P_1, P_2, ..., P_n, i.e., some of the classes in P are distributed among the new packages and after that we have a restructured package P', with fewer classes. We analyzed all 21 distinct module decompositions (in Table IV one decomposition can impact more than one cluster) in the dataset under two dimensions:

- Paired Comparisons: We compared the original package P with its restructured version P'. Figure 5 reports two boxplots. The first one (named Before) represents the conceptual cohesion of the original packages P of the collected module decompositions. Similarly, the second boxplot represents the conceptual cohesion of the restructured packages P'. We observe an improvement in most packages.
- New Packages: We compared the average cohesion of the new packages P_1, P_2, ..., P_n with the original package P, as presented by the boxplot New Packages in Figure 5. An improvement in such measure means that the classes are typically more cohesive in the new packages than in the original one. We also observed that 0.40 is a good target cohesion measure to achieve after module decompositions. Most of the packages we investigated have their conceptual cohesion above this measure.

TABLE IV
MODULARIZATION OPERATORS RESPONSIBLE FOR THE TOP-3 AND THE BOTTOM-3 CHANGES IN SPREAD AND FOCUS (MD= MODULE DECOMPOSITION; MU= MODULE UNION; FT= FILE TRANSFERAL; DT= DATA STRUCTURE TRANSFERAL; RN= RENAME; PF= PROMOTE FUNCTION; MR= MODULE REMOVAL; FR= FILE REMOVAL)

Metric	Ranking	System	Cluster id	Cluster Definition	Operators
Spread	Top 3	Eclipse 2.1→3.0	105	Layout data structures	MD
			14	Workbenchs status and configuration	MD, FT, DT
			61	Command classes, undo and rewrite managers	
		JHotDraw 7.3.1→7.4.1	27	Connection Figures	MD
			34	File and text edition actions	MD
			40	Text undo and selection actions	MD
		Next 12-08-07→12-11	26	Data report	MD
			25	Special data types, SQL translators	
			12	Database operators, Javascript builders	MD
		Vivo 1.4.1→1.5	18	Individual management	MD
			16	Page and HTML requests and servlets	MD
			29	Servlet and request exceptions	MD
	Bottom 3	Eclipse 2.1→3.0	101	Text edition messages and actions	FT, DT
			54	Marker classes	MD
			38	JFace's viewers, listeners and element data	
		JHotDraw 7.3.1→7.4.1	17	XML and IO utilities	
			38	Color Representation	MR
		Next 12-08-07→12-11	1	Bean data validation	MD,FT
			3	Core loaders	FR
			8	UI's input components	
		Vivo 1.4.1→1.5	33	Action permission	RN, FR
			10	Web ontology management and permissions	MU
			5	Authorization and permission actions	MU, MD
Focus	Top 3	Eclipse 2.1→3.0	51	Help queries and search results	MU, MD
			33	Registry for marker help contexts	MD, FT
			12	Annotation models	MD, PF
		JHotDraw 7.3.1→7.4.1	34	File and text edition actions	MD
			12	Figure manipulation	MD
			16	Line Drawing	MD
		Next 12-08-07→12-11	14	Core loaders	MD
			9	Properties and String manipulation	
			22	Loaders management	MD
		Vivo 1.4.1→1.5	18	Individual management	MD
			9	User authentication	
			22	Content filtering	MD
	Bottom 3	Eclipse 2.1→3.0	39	UI's widget events	FT, DT
			94	Command's data structures	MD
			6	Action classes	
		JHotDraw 7.3.1→7.4.1	20	Font chooser and preferences management	
			23	Palette functions	
			32	Drawing element attributes management	
		Next 12-08-07→12-11	4	Report aggregating functions	RN
			21	HTTP requests	
			16	Class compiling and loading	RN, MU
		Vivo 1.4.1→1.5	32	N3 edition	FR
			13	Data and object property permissions	RN
			28	Converter classes, Query generation	

As well as for the global results, JHotDraw 7.3.1→7.4.1 presented the best results in both comparisons. For example, the package `draw` had the best improve in conceptual cohesion (+0.111); moreover its ten new subpackages have an average cohesion of 0.818, which is also a very good measure. A similar case occurred in Eclipse, in which a decomposition in the `ui.ide` plugin involved class transferal to the packages in the `ui.workbench` plugin. As a result, these packages increased their cohesion in 0.016 and 0.048, respectively.

Vivo was the only system with a decrease in conceptual cohesion after module decompositions. For example, the new `dataGetter` package is less cohesive than its original package, `pageDataGetter`. This fact is explained by the use of a new interface in the restructured package, which has more utility classes and therefore a larger vocabulary. Thus, the similarity between the classes changed to use this new interface and the rest of the package has decreased.

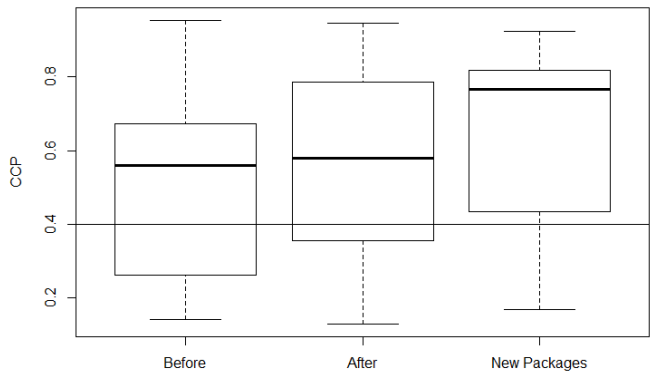


Fig. 5. Average Conceptual Cohesion Results using CCP

Summary: After module decompositions, the new packages have better conceptual cohesion than the original packages. CCP is an adequate metric to express a quality improvement, but some care must be taken in interpretation (e.g., the Vivo case). Typically, packages created after module decompositions have internal cohesion greater than 0.40.

VIII. Threats to Validity

This section presents the limitations that can affect the results of our work. As any information retrieval technique, Semantic Clustering relies on vocabulary quality. Thus, we assume that terms are well described in the system's identifiers and comments. Naming conventions other than *camelCase* and *under_score* will certainly compromise the vocabulary. The representativeness of the terms in the vocabulary can also impact on our results. We identified in previous studies the existence of methods like "kaboom" or "nothing". In this case, the vocabulary is polluted with terms that do not often appear and also do not have proper meaning. In this work, we selected systems with a considerable developer community. Therefore, we consider these issues an exception in the vocabulary.

Concerning the remodularization analysis, we consider our manual operations in package removal (Section VII-B) and modularization operators identification (Section VII-D), a thread to this study. We recognize that these operations are not easy to automate. They also require an advanced knowledge of each system, being a time spending task to perform manually. However, we did make an effort to follow a methodology, and we also tried to isolate the remodularizations from bug corrections and enhancements, in order to gather convincing results from our study with conceptual metrics.

IX. Related Work

We organized related work in three groups: (i) Evaluation of Structural Metrics; (ii) Conceptual Metrics; (iii) Application of Information Retrieval Techniques in Reverse Engineering.

Evaluation of Traditional Metrics: Regarding architectural quality, most of quality metrics consider static and structural aspects, like method calls or use of attributes. For example, Abreu and Goulão proposed a procedure do improve modularity using clustering and coupling metrics [4]. They observed that the improved modularization differed from the original one in the number of modules. Therefore, they concluded that practitioners do not seem to use cohesion and coupling as driving forces when it comes to modularization.

Moreover, there is a large variety of metrics that attempt to measure the same aspect of quality. Ó Cinneide *et al.* describe a comparative evaluation of five widely used, structural cohesion metrics [5]. The goal was to verify whether the value of these metrics evolve together, given a same refactoring action. Considering eight systems and over three thousand refactorings, in 38% of the cases one metric value increased while the other value decreased. This fact is an indication that the traditional cohesion metrics measure different and conflicting aspects of the same property.

Concerning real remodularization cases, Anquetil and Laval used structural cohesion and coupling metrics over three remodularizations of Eclipse, in order to verify whether the metrics follow the widely recommended quality guideline of high cohesion and low coupling [2]. However, the coupling values increased in most of the packages, and the cohesion metric presented a flaw in the experiments. They concluded that either the structural metrics or the cohesion/coupling dogma fails in representing architectural quality.

Conceptual Metrics: On the other hand, a new set of metrics has been proposed in recent work by Poshyvanyk *et al.* [6, 7, 23]. Conceptual metrics were evaluated by correlating them with number of faults, and in comparison to existing structural metrics. Most of them were able to predict faults in classes. However, they did not use their metrics to evaluate architectural quality, and particularly to analyze the benefits of remodularizations. In this work, we selected the simpler metric (Conceptual Cohesion of a Class) of the set.

Application of Information Retrieval Techniques in Reverse Engineering: Anquetil and Lethbridge [24] proposed one of the first approaches to extract clusters from entity names. They extracted information (*n-grams*) from file names, obtaining clusters of files with *n-grams* in common. In comparison with other clustering techniques, they concluded that the file name criterion is more likely to discover subsystems in a legacy system. Maletic and Marcus were among the first to propose the use of LSI to extract clusters and also propose the comparison between the system's structure and the generated clusters, at the level of procedures [25, 26].

Recent work in architecture recovery comprise the combination of structural and conceptual aspects. Scanniello *et. al* [27] proposed the use of structural links to derive architectural layers. Lexical information is then analyzed to decompose each layer into modules. Thus, module decomposition is derived through clusters.

In a simpler approach, Bavota *et al.* proposed to identify Module Decomposition opportunities [15, 28]. Given one module, chains of classes with high coupling are found, and a new package is proposed for each chain. Coupling is measured by a combination of two kinds of metrics: structural and conceptual. They applied the approach in five systems,

including JHotDraw, and concluded that the refactoring suggestions are meaningful from the perspective of developers. Our work complements this approach by reinforcing that module decompositions are the operations with the highest impact on semantic clusters.

X. Conclusion and Future Work

A common criticism raised on the disuse of quality metrics is that they lack a proper validation. In this work, we presented the evaluation of conceptual metrics regarding real remodularization cases. We propose the use of Semantic Clustering and conceptual metrics to check whether developers consider the vocabulary of entities when reorganizing a system's architecture. In our study with four systems and six remodularizations, we gathered indicatives that state the consequences of applying Module Decomposition. Conceptual metrics were able to describe an improvement in most of the cases in which this operation was performed. In general, the creation of highly cohesive packages comes at a price of increasing the concept spread over new packages. This fact reveals that developers organize the system's classes in packages according to a common intent, or concept. Our findings encourage us to extend this study at the point of designing a tool to recommend (re-)modularization operations based on conceptual metrics.

Acknowledgments

Our research is supported by CAPES, FAPEMIG, and CNPq.

References

[1] L. Passos, R. Terra, R. Diniz, M. T. Valente, and N. C. Mendonca, "Static architecture conformance checking – an illustrative overview," *IEEE Software*, vol. 27, no. 5, pp. 82–89, 2010.

[2] N. Anquetil and J. Laval, "Legacy software restructuring: Analyzing a concrete case," in *15th European Conference on Software Maintenance and Reengineering*, pp. 279–286, 2011.

[3] S. Mancoridis, B. S. Mitchell, Y. Chen, and E. R. Gansner, "Bunch: A clustering tool for the recovery and maintenance of software system structures," in *15th International Conference on Software Maintenance*, pp. 50–, 1999.

[4] F. B. Abreu and M. Goulão, "Coupling and cohesion as modularization drivers: Are we being over-persuaded?," in *5th European Conference on Software Maintenance and Reengineering*, pp. 47–, 2001.

[5] M. Ó Cinnéide, L. Tratt, M. Harman, S. Counsell, and I. Hemati Moghadam, "Experimental assessment of software metrics using automated refactoring," in *International Symposium on Empirical Software Engineering and Measurement*, pp. 49–58, 2012.

[6] A. Marcus and D. Poshyvanyk, "The conceptual cohesion of classes," in *21st International Conference on Software Maintenance*, pp. 133–142, 2005.

[7] B. Ujhazi, R. Ferenc, D. Poshyvanyk, and T. Gyimóthy, "New conceptual coupling and cohesion metrics for object-oriented systems," in *10th International Working Conference on Source Code Analysis and Manipulation*, pp. 33–42, 2010.

[8] S. L. Abebe, S. Haiduc, A. Marcus, P. Tonella, and G. Antoniol, "Analyzing the evolution of the source code vocabulary," in *13th European Conference on Software Maintenance and Reengineering*, pp. 189–198, 2009.

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

[10] A. Kuhn, S. Ducasse, and T. Gîrba, "Enriching reverse engineering with semantic clustering," in *12th Working Conference on Reverse Engineering*, pp. 133–142, 2005.

[11] S. Ducasse, T. Gîrba, and A. Kuhn, "Distribution map," in *22nd International Conference on Software Maintenance*, pp. 203–212, 2006.

[12] G. M. Rama and N. Patel, "Software modularization operators," in *26th International Conference on Software Maintenance*, pp. 1–10, 2010.

[13] M. Fowler, *Refactoring: Improving the Design of Existing Code*. Boston, MA, USA: Addison-Wesley, 1999.

[14] D. Beyer and A. Noack, "Clustering software artifacts based on frequent common changes," in *13th International Workshop on Program Comprehension*, pp. 259–268, 2005.

[15] G. Bavota, A. D. Lucia, A. Marcus, and R. Oliveto, "Using structural and semantic measures to improve software modularization," *Empirical Software Engineering*, vol. 18, no. 5, pp. 901–932, 2013.

[16] R. Terra and M. T. Valente, "A dependency constraint language to manage object-oriented software architectures," *Software: Practice and Experience*, vol. 32, no. 12, pp. 1073–1094, 2009.

[17] G. Salton and M. J. McGill, *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., 1986.

[18] D. Poshyvanyk, M. Gethers, and A. Marcus, "Concept location using formal concept analysis and information retrieval," *ACM Transactions on Software Engineering and Methodology*, vol. 21, no. 4, p. 23, 2012.

[19] M. Gethers, T. Savage, M. D. Penta, R. Oliveto, D. Poshyvanyk, and A. D. Lucia, "Codetopics: which topic am I coding now?," in *33rd International Conference on Software Engineering*, pp. 1034–1036, 2011.

[20] R. A. Baeza-Yates and B. A. Ribeiro-Neto, *Modern Information Retrieval, Second edition*. Pearson Education Ltd., 2011.

[21] K. S. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, "When is "nearest neighbor" meaningful?," in *7th International Conference on Database Theory*, pp. 217–235, 1999.

[22] C. Taube-Schock, R. J. Walker, and I. H. Witten, "Can we avoid high coupling?," in *25th European conference on Object-oriented programming*, pp. 204–228, 2011.

[23] D. Poshyvanyk and A. Marcus, "The conceptual coupling metrics for object-oriented systems," in *22nd International Conference on Software Maintenance*, pp. 469–478, 2006.

[24] N. Anquetil and T. Lethbridge, "File clustering using naming conventions for legacy systems," in *1997 Conference of the Centre for Advanced Studies on Collaborative research*, pp. 2–, 1997.

[25] J. I. Maletic and A. Marcus, "Supporting program comprehension using semantic and structural information," in *23rd International Conference on Software Engineering*, pp. 103–112, 2001.

[26] A. Marcus, A. Sergeyev, V. Rajlich, and J. I. Maletic, "An information retrieval approach to concept location in source code," in *11th Working Conference on Reverse Engineering*, pp. 214–223, 2004.

[27] G. Scanniello, A. D'Amico, C. D'Amico, and T. D'Amico, "Using the kleinberg algorithm and vector space model for software system clustering," in *18th International Conference on Program Comprehension*, pp. 180–189, 2010.

[28] G. Bavota, A. D. Lucia, A. Marcus, and R. Oliveto, "Software re-modularization based on structural and semantic metrics," in *17th Working Conference on Reverse Engineering*, pp. 195–204, 2010.