Hindawi
Scientific Programming
Volume 2017, Article ID 3787053, 15 pages
https://doi.org/10.1155/2017/3787053

Research Article

Clustering Classes in Packages for Program Comprehension

Xiaobing Sun,[1,2] Xiangyue Liu,[1] Bin Li,[1] Bixin Li,[3] David Lo,[4] and Lingzhi Liao[5]

[1]*School of Information Engineering, Yangzhou University, Yangzhou, China*
[2]*State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China*
[3]*School of Computer Science and Engineering, Southeast University, Nanjing, China*
[4]*School of Information Systems, Singapore Management University, Singapore*
[5]*Nanjing University of Information Science & Technology, Nanjing, China*

Correspondence should be addressed to Bin Li; lb@yzu.edu.cn

Received 16 October 2016; Revised 13 February 2017; Accepted 27 February 2017; Published 11 April 2017

Academic Editor: Xuanhua Shi

During software maintenance and evolution, one of the important tasks faced by developers is to understand a system quickly and accurately. With the increasing size and complexity of an evolving system, program comprehension becomes an increasingly difficult activity. Given a target system for comprehension, developers may first focus on the package comprehension. The packages in the system are of different sizes. For small-sized packages in the system, developers can easily comprehend them. However, for large-sized packages, they are difficult to understand. In this article, we focus on understanding these large-sized packages and propose a novel program comprehension approach for large-sized packages, which utilizes the Latent Dirichlet Allocation (LDA) model to cluster large-sized packages. Thus, these large-sized packages are separated as small-sized clusters, which are easier for developers to comprehend. Empirical studies on four real-world software projects demonstrate the effectiveness of our approach. The results show that the effectiveness of our approach is better than Latent Semantic Indexing- (LSI-) and Probabilistic Latent Semantic Analysis- (PLSA-) based clustering approaches. In addition, we find that the topic that labels each cluster is useful for program comprehension.

1. Introduction

Program comprehension is one of the most frequently performed activities in software maintenance [1, 2]. It is a process whereby a software practitioner understands a program using both knowledge of the domain and semantic and syntax knowledge, to build a mental model of the program [3, 4]. Developers working on software maintenance tasks spend around 60% of their time for program comprehension [5]. As software evolves, its complexity becomes increasingly higher. Moreover, some documents affiliated to the system also become outdated or inaccessible, which makes program comprehension more difficult.

In practice, the natural top-down program comprehension process can effectively facilitate developers to understand the system step by step [6]. For an object oriented Java software system, developers can also understand a system in such a top-down way. Packages are first taken into consideration. Then, interesting packages are selected, and developers further go deep into classes in the packages. For small-sized packages (with several classes), it is easy for developers to understand them. However, for packages with many classes in them, it is more challenging for developers to understand these classes, their relationships, and their functionalities [7, 8]. To aid program understanding, classes in these large-sized packages can be clustered into smaller-sized groups. With such clustering, developers can understand a system more easily.

There are several approaches that cluster programs based on static structural dependencies in the source code [9]. Static structural dependencies based approaches usually cluster classes in a system based on static structural dependencies among program elements, such as variable and class references, procedure calls, use of packages, and association and inheritance relationships among classes [8, 10, 11]. These approaches are more suitable in the process of implementing a change request in the source code. But before implementing a change request in the source code, developers should know

which part in the source code is related to the change request. Specifically, they need to know the functional points of a system and where in the source code corresponds to these functional features. A feature or functional point represents a functionality that is defined by requirements and accessible to developers and users. Then, they can implement source code level changes. Hence, some studies focused on understanding the functional features of a system and proposed semantic clustering, which exploits linguistic information in the source code, such as identifier names and comments [12]. These approaches usually take the whole system as input and generate clusters at some granularity levels, for example, class level or method level. The generated clusters corresponding to different functional features are used to divide a system into different units [13, 14]. This article also focuses on exploiting linguistic information in the source code to understand functional features of different clusters in large-sized packages. In a large-sized package, there are a number of functional features or concerns. Each of these concerns is implemented in a set of classes. The previous studies focused on clustering a software unit. However, developers still do not easily know what the functional features that each cluster expresses are. So to get a good understanding of its concerns and the classes that implement each of them, in this article, we further generate a set of topics to describe each cluster.

This article proposes a technique to generate a set of clusters of classes for a large-sized package, where different clusters correspond to different functional features or concerns. Our approach is based on Latent Dirichlet Allocation (LDA), which is a topic model and one of the popular ways to analyze unstructured text in the corpus [15]. LDA can discover a set of ideas or themes that well describe the entire corpus. We use LDA for a whole package and extract latent topics to capture its functional features. Then, classes in the package with similar topics are clustered together.

Our approach can be effectively used for top-down program comprehension during software maintenance. For small-sized packages, developers can directly understand them. For large-sized packages, our approach can be used to divide packages into small clusters. Each of these small clusters can be understood more easily than the original large-sized package. The main contributions of this article are as follows:

(1) We propose to use LDA to generate clusters for large-sized packages. The topics generated by LDA are useful to indicate the functional features for these class clusters.

(2) We conduct an empirical study to show the effectiveness of our approach on four real-world open-source projects, *JHotDraw*, *jEdit*, *JFreeChart,* and *muCommander*. The results show that our approach is more effective in identifying more relevant classes in the cluster than other semantic clustering approaches, that is, Latent Semantic Indexing- (LSI-) and Probabilistic Latent Semantic Analysis- (PLSA-) based clustering.

(3) The empirical study on four selected packages from four subject systems shows that the topics generated by our approach are useful to help developers understand these packages.

The rest of the article is organized as follows: in the next section, we introduce the background of program comprehension and LDA model. Section 3 describes our approach. We describe the design of our experiment, experiment results, and threats to validity of our study in Sections 4, 5, and 6, respectively. In Section 7, related work using clustering for program comprehension is discussed. Finally, we conclude the article and outline directions for future work in Section 8.

2. Background

In this article, we use LDA to cluster classes in large-sized packages for easier program comprehension. This section discusses the background of program comprehension and LDA topic model.

2.1. Program Comprehension. For software developers, program comprehension is a process whereby they understand a software artifact using both knowledge of the domain and semantic and syntax knowledge [10]. Program comprehension can be divided into bottom-up comprehension, top-down comprehension, and various combinations of these two processes. In bottom-up comprehension, a developer may first read all the source code at finer statement or method level and abstract features and concepts according to the low-level information. Then, coarser class-level or package-level elements are read and understood. Finally, developers comprehend the whole system. For top-down comprehension, a developer first utilizes knowledge about the domain to build a set of expectations that are mapped to the source code. Then, he/she understands the coarser package-level or class-level elements, followed by finer method-level or statement-level elements. Finally, the developer also gets an understanding of the whole system. In practice, top-down program comprehension is more acceptable since it meets humans' way of thinking from simple to complex, from whole to part [3].

Software clustering is one of the effective techniques for top-down program comprehension. During software maintenance, developers usually need to identify the functional features they are interested in to help them accomplish a change request. In this article, we propose a software clustering technique using LDA to provide some features for developers to facilitate the top-down program comprehension process.

2.2. Latent Dirichlet Allocation. Topic models were originated from the field of information retrieval (IR) to index, search, and cluster a large amount of unstructured and unlabeled documents. A topic is a collection of terms that cooccur frequently in the documents of the corpus. One of the mostly used topic models in software engineering community is Latent Dirichlet Allocation (LDA) [16–18]. It requires no training data and can well scale to thousands or millions of documents.

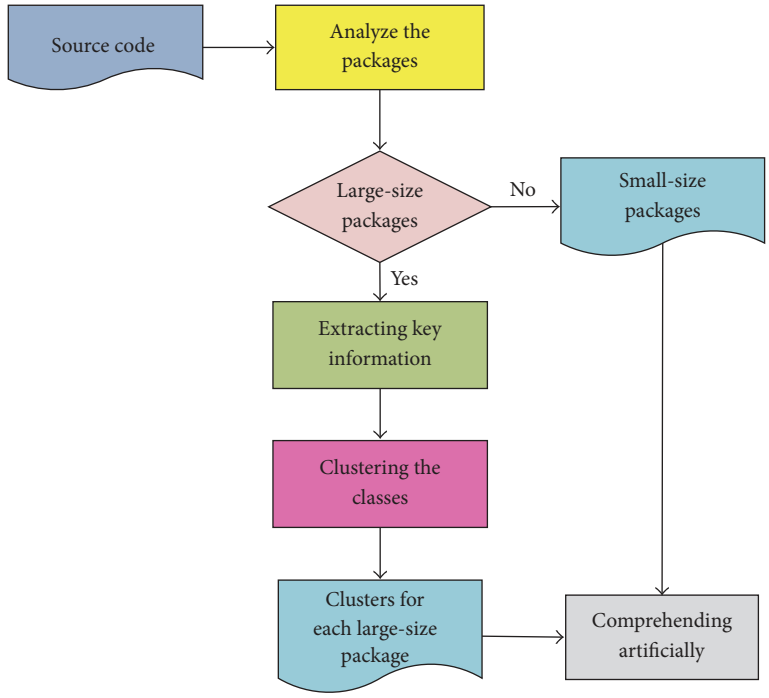


Figure 1: Process of our approach.

LDA models each document as a mixture of K corpus-wide topics and each topic as a mixture of terms in the corpus [15]. More specifically, it means that there is a set of topics to describe the entire corpus; each document can contain more than one of these topics; and each term in the entire repository can be contained in more than one of these topics. Hence, LDA is able to discover a set of ideas or themes that well describe the entire corpus. It assumes that documents have been generated using the probability distribution of the topics and that words in the documents were generated probabilistically in a similar way.

In order to apply LDA to the source code, we represent a software system as a collection of documents (i.e., classes) where each document is associated with a set of concepts (i.e., topics). Specifically, the LDA model consists of the following building blocks:

(1) A word is the basic unit of discrete data, defined to be an item from a software vocabulary $V = \{w_1, w_2, \ldots, w_v\}$, such as an identifier or a word from a comment.

(2) A document, which corresponds to a class, is a sequence of words denoted by $d = \{w_1, w_2, \ldots, w_n\}$, where w_i is the ith word in the sequence.

(3) A corpus is a collection of documents (classes) denoted by $D = \{d_1, d_2, \ldots, d_m\}$.

Given m documents containing k topics expressed over v unique words, the distribution of ith topic t_i over v words and the distribution of jth document over k topics can be represented.

By using LDA, it is possible to formulate the problem of discovering a set of topics describing a set of source code classes by viewing these classes as mixtures of probabilistic topics. For further details on LDA, interested readers are referred to the original work of Blei et al. [15].

With LDA, latent topics can be mined, allowing us to cluster them on the basis of their shared topics. In this article, to effectively use LDA, we apply it in a package-level corpus rather than each class to extract the latent topics to simulate the functional features or concerns for a package since small (class-level) corpus is too small to generate good topics [19–23]. Then, we cluster the classes according to these topics and assign different classes to their corresponding topics [23].

3. Our Approach

Faced with the source code of a software system, developers need to use their domain knowledge to comprehend the program from coarse package level to class level in each package. The process of understanding different packages is different. In the process, small-sized packages are easy to understand while large-sized packages are complex and they need to be separated into small-sized clusters. In this article, we focus on clustering the classes in large-sized packages as well as their corresponding functional features.

The process of understanding packages is described in Figure 1. Firstly, we analyze the size of each package in the software system. Small-sized packages are comprehended manually. For large-sized packages, there are two steps. First, LDA is used to extract the latent key information to facilitate the comprehension process. Then, on the basis of the key information of each package, we adopt the clustering to build small-sized clusters to decompose each package. Thus, given the source code of a software system at hand, programmers

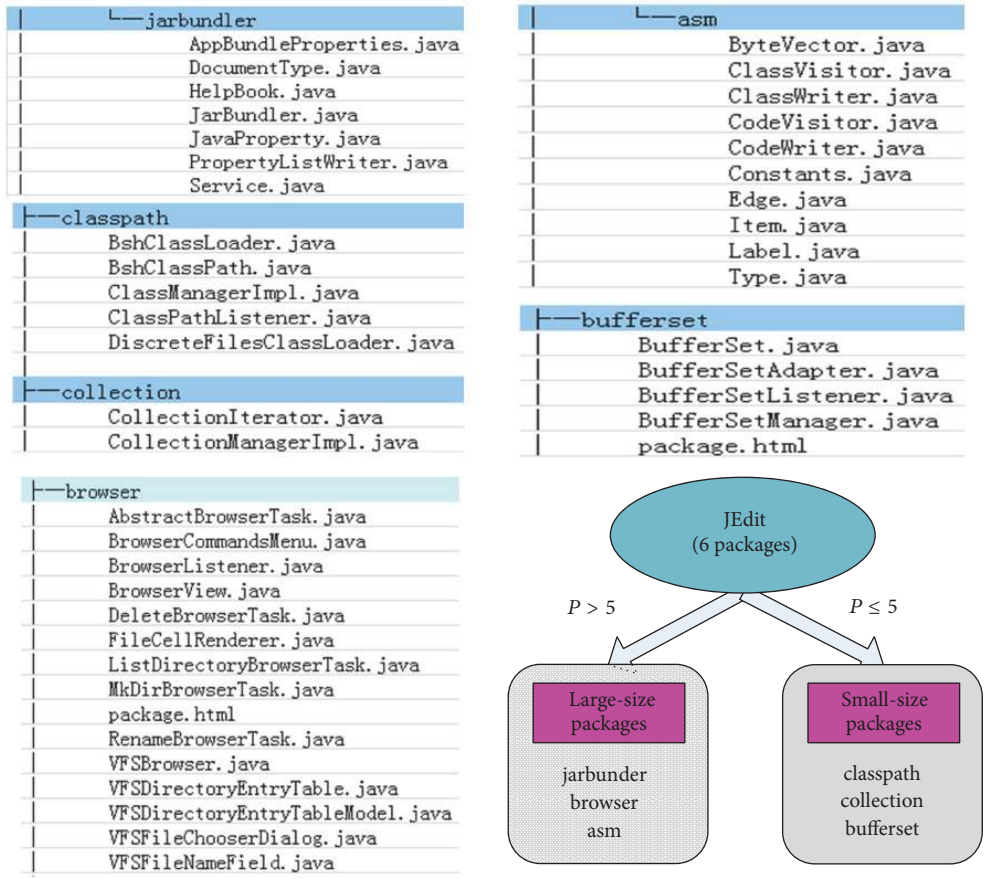


Figure 2: An example of separating packages into large-sized packages and small-sized packages for six packages in jEdit when P is set to 5.

can comprehend small-sized packages by themselves and large-sized packages with the help of our approach. In the following subsections, we discuss more details of our approach.

3.1. Analyzing the Size of Packages. Our approach focuses on understanding large-sized packages. So we first need to select large-sized packages in a program. Here, we set a threshold P for the number of classes in a package. The packages including more than P classes are selected for analysis. These packages are separated into smaller clusters to facilitate program comprehension. Figure 2 shows an example of separating packages into large-sized packages and small-sized packages for six packages in *jEdit* when P is set to 5. The packages *jarbunder*, *browser,* and *asm* are classified as large-sized packages.

3.2. Extracting Key Information Based on LDA. During the program comprehension process, developers are more focused on functional features or concerns of the program. In the program, the source code contains not only the syntax information but also the unstructured data, such as natural language identifiers and comments [24]. These unstructured source code identifiers and comments can be used to capture the semantics of the developers' intent [25]. They represent an important source of domain information and can often serve as a starting point in many program comprehension tasks [26, 27]. However, there exists noise in the source code, which can potentially confuse the LDA application. So natural language processing (NLP) techniques are usually used to perform one or more preprocessing operations before applying LDA models to the source code data. Then, LDA can be effectively used to generate the topics. To effectively use LDA, we apply it in a package-level corpus to simulate the functional features or concerns for a package. Finally, we cluster the classes according to these topics and assign different classes to their corresponding topics.

3.2.1. Preprocessing of the Source Code. There are several typical preprocessing operations for the unstructured part of a source code. These preprocessing operations can be performed to reduce noise and improve the quality of the resulting text for LDA [28].

We first isolate identifiers and comments and strip away syntax and programming language keywords (e.g., "*public*" and "*int*"). First, we remove header comments since they often include generic information about the software that are included in most of source code files. Then, we tokenize each word based on common naming practice, such as camel

Figure 3: The process of preprocessing the class *InvalidHeaderException.java* in jEdit.

case *("oneTwo")* and underscores *("one_two")*, and remove common English language stop words *(the*, *it*, and *on)* and programming language related key words *(public*, *int*, and *while)*.

After preprocessing the unstructured part of source code files, LDA can be used to extract key information more effectively. Figure 3 shows an example of the detailed process of preprocessing the source code in the class *InvalidHeaderException.java* in *jEdit*. After preprocessing the source code, most of the useful words are kept for LDA application.

3.2.2. Extracting Key Information from Large-Sized Packages. After preprocessing each class in large-sized packages, we need to extract key information from them. LDA is an effective approach to discover a set of ideas or themes that well describe the entire corpus. Before using LDA, we need to set the number of topics, that is, K. This parameter affects the effectiveness of LDA application. In this article, K is related to the size of clusters for a package, which is determined by users.

An LDA application generates two files: one is the word-topic matrix which lists the words for each topic and the other is the topic-document matrix which shows the percentage of topics in each document, also called the membership value. An example is given in Figure 4. The results show the distribution of different topics in different classes, and each topic is described by different words with different possibilities. These topics express different functional features of classes in the package.

3.3. Clustering Classes in Large-Sized Packages. After extracting the topics from the objective package, classes having similar topics should be allocated in a cluster to aid their comprehension. In this subsection, we discuss details for clustering classes in a package.

3.3.1. Generating Initial Clusters. To cluster classes in a package, the number of clusters should be first determined. However, it is difficult to know this information at the

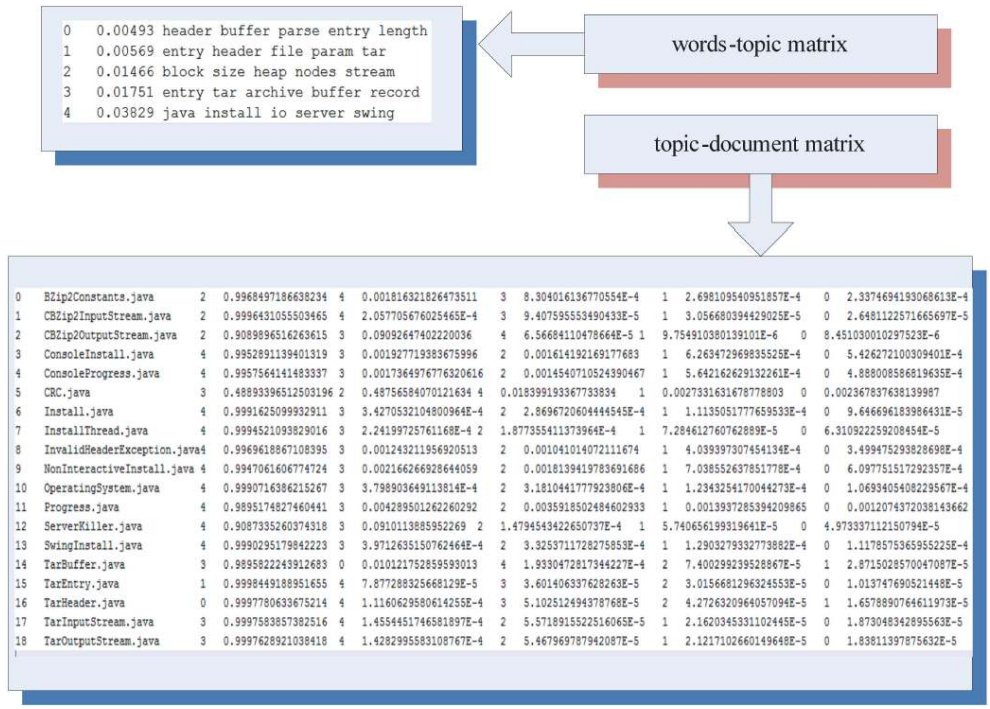


Figure 4: An example of the output of an LDA application.

beginning. In this article, the number of clusters is estimated based on the number of classes in a cluster.

Let us assume that the number of classes in an initial cluster is M, where M is a user-defined parameter. That is, if a user thinks that an M-scale cluster is easy for him/her to understand, he/she can set the initial size of each cluster as M, for example, 5 and 10. For a package with N classes ($N \geq M$), each of these classes should be put into a cluster. Thus there will be $\lceil N/M \rceil$ (a whole number) clusters for a package. We set the number of topics K to be the same as the number of clusters (i.e., $\lceil N/M \rceil$), because we need a topic to label each cluster.

After applying the LDA in a preprocessed package, we get two files, the word-topic matrix which lists the words for each topic and the topic-document matrix which shows the percentage of the topic words in each document. To allocate different classes to their corresponding topics, we use the topic-document matrix. That is, we allocate the top M documents to these K topics in the topic-document matrix. Thus a set of clusters can be generated, which we call the *initial clusters* for a package.

The ideal situation for the initial clusters is that each class is just assigned to only its own and exclusive cluster. Inevitably, there are two special cases; one is that a class may match different topics in the topic-document matrix. Such classes are called shared classes, which we need to reassign. The other case is that there may be some remaining classes that are not included in the top M documents in any topics. Such classes are called nonmatching classes, which need to be assigned to the most probable clusters related to them. In the following, we deal with these classes to guarantee that each class is assigned to one and only one cluster.

3.3.2. Assigning Shared Classes and Nonmatching Classes. Shared classes are the classes matching different topics in the generated topic-document matrix. These classes are all listed in the top M classes for each topic. We list all the classes shared by different topics and the membership value of each topic for them. A shared class is allocated to the cluster corresponding to the topic with the highest membership value.

For nonmatching classes that are not initially matched to any cluster, they are processed in a similar way as shared classes. We list all these nonmatching classes and their membership values. Each of these nonmatching classes is put into the cluster corresponding to the topic with the highest membership value.

Finally, each cluster in a package only contains classes having high membership values and each class is a member of only one cluster. Based on the word-topic matrix, we can see the words describing the topic, which indicates the feature of the cluster. Figure 5 shows an example of the process to generate the clusters for a large-sized package. First, initial clusters are generated according to the membership values with five topics. Then, shared classes and nonmatching classes are assigned to corresponding initial clusters based on their membership values. Finally, a set of clusters for the large-sized package is obtained as shown in the bottom-left part of Figure 5.

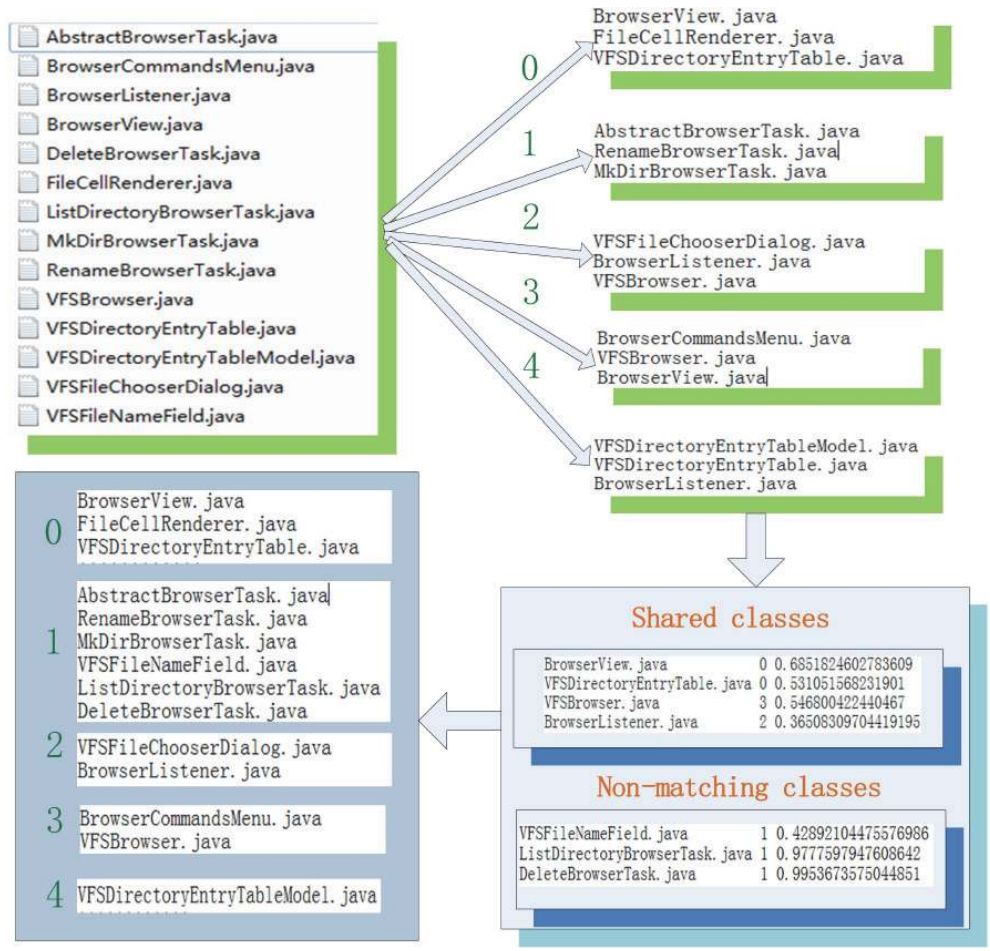


Figure 5: An example of generating clusters for a large-sized package (with the number of topics set to 5).

4. Case Study

In this section, we conduct case studies to evaluate the effectiveness of our approach. In our study, we address the following three research questions *(RQs)*:

> *RQ1*: Does the number of topics affect the shared classes and nonmatching classes?
>
> *RQ2*: Is our LDA clustering approach more effective than other semantic clustering approaches, that is, LSI-based clustering and PLSA-based clustering?
>
> *RQ3*: Can our approach provide useful topics for developers to understand the classes in the package(s)?

In our approach, the number of topics is set by users themselves. We investigate *RQ1* to see how this parameter affects the number of shared and nonmatching classes. In addition, we investigate *RQ2* to see whether our clustering approach using LDA is more effective than other semantic clustering based approaches based on LSI and PLSA [12, 29–31], respectively. Finally, there is another difference between our approach and other clustering approaches; that is, each cluster that is generated by our approach is labeled with a topic to facilitate understanding of the cluster. So *RQ3* aims to answer whether the topic labeling each cluster can help developers understand the cluster or not.

4.1. Empirical Environment. We implemented our approach with Java language in the Eclipse environment. In addition, all the selected subject programs are also Java programs. So our case study was conducted in the Eclipse environment.

4.2. Subject Systems. We address our research questions by performing case studies on the source code of four well-known software systems, *JHotDraw* (https://sourceforge.net/projects/jhotdraw), *jEdit* (https://sourceforge.net/projects/jedit), *JFreeChart* (https://sourceforge.net/projects/jfreechart), and *muCommander* (http://www.mucommander.com), as shown in Table 1.

JHotDraw is a medium-sized, open-source, 2D drawing framework developed in the Java programming language. *jEdit* is a medium-sized, open-source text editor written in Java. *JFreeChart* is a free 100% Java chart library that makes it easy for developers to display professional quality charts

Table 1: Subject systems.

Subject	Version	Files	Packages	Classes
JHotDraw	7.0.6	144	23	305
jEdit	5.1.0	147	43	573
JFreeChart	1.0.17	105	70	990
muCommander	0.9.0	98	89	692

Table 2: The percentage of classes over P (5, 10, and 15) of the four systems.

Subject	$P > 5$ (%)	$P > 10$ (%)	$P > 15$ (%)
JHotDraw	91.8	79.7	68.5
jEdit	91.1	81.3	72.2
JFreeChart	93.2	81.0	67.6
muCommander	81.1	56.9	37.2
Average	89.3	74.7	61.4

in their applications. *muCommander* is a lightweight, cross-platform file manager that runs on any operating system supporting Java.

These projects belong to different problem domains. They are general enough to represent real-world software systems, and they have been widely used in empirical studies in the context of software maintenance and evolution [32, 33]. In addition, they have become the de facto standard system for experiments and analysis in topic and concern mining (e.g., by Robillard and Murphy [34] and Binkley et al. [35]). Moreover, these four subject systems of different sizes that are neither too small nor too large are selected due to their good design and manageable size for manual analysis.

4.3. Parameters Setting. In our approach, there are two parameters, P and K. P represents the size of a package and K is the number of topics as input for the LDA model. Values of these two parameters will affect the number of packages to be subdivided into clusters and the number of clusters in a package. Table 2 shows the percentage of classes in packages with different number of classes. From the results, when P is 5, 10, and 15, the average percentages of classes in packages of large sizes are 89.3%, 74.7%, and 61.4%, respectively. In this study, we consider packages of size larger than 10 as the large-sized packages used to evaluate our approach. Hence, for all four systems, most of classes and packages are used to evaluate our approach.

The other parameter in our approach is the number of topics (K) for LDA analysis. K is an important parameter which also indicates the number of clusters for the final clustering results. It determines the size of each cluster. We set K to be 5, 10, and 15 for our study, respectively.

4.4. Methods and Measures. For LDA computation, we used *MALLET* (http://mallet.cs.umass.edu), which is a highly scalable Java implementation of the *Gibbs* sampling algorithm. We ran for 10,000 sampling iterations, the first 1000 of which were used for parameter optimization. We selected different numbers of topics to use *MALLET* to generate the word-topic matrix and topic-document matrix. Then, we clustered each large-sized package based on these two matrixes.

For semantic clustering based on LSI and PLSA [12, 29–31] that we used to compare with our approach, they are popular methods for cluster analysis, especially for clustering nonstructured data. LSI uses singular value decomposition to explore patterns in the relationships between the terms and concepts contained in an unstructured corpus [36]. LSI is implemented based on the assumption that words used in the same contexts tend to have similar meanings. Hence, LSI is able to extract the conceptual contents from a corpus by establishing associations between those terms that occur in similar contexts. Probabilistic Latent Semantic Analysis (PLSA) is a statistical technique for the data analysis, which is based on a mixture decomposition derived from a latent class model [30, 31].

We selected these clustering approaches for comparison because (1) they are widely used in clustering software data and show promising results [37, 38] and (2) they are also clustering approaches based on lexical information which is similar to our approach. In our study, they are performed by clustering classes with similar vocabularies. After calculating the similarity between each pair of documents, an agglomerative hierarchical clustering algorithm is executed. There are a lot of similarity measures, for example, cosine similarity, Manhattan distance, and Euclidean distance [39]. Cosine similarity which is a popular similarity measure is used here [33, 40].

To answer *RQ1*, we compute the number of shared classes and nonmatching classes and the *shared occurrence counts* (or *shared counts*) of the shared classes. For example, if one class is shared by topic 1 and topic 2, its shared count is 1. If the class is also shared by topic 3, the shared count is 2. We analyze the percentages of shared classes and nonmatching classes as well as the shared counts for different numbers of topics.

For *RQ2*, our study involved 10 participants from university and industry. Half of them are from our lab with 2-3 years of development experience and the other half are from industry with 5-6 years of development experience especially

Table 3: The selected packages and selected classes.

Subject	Package	Size	Class	K
JHotDraw	*JHotDraw.src.org.jhotdraw.app.action*	33	*AbstractProjectAction.java*	10
jEdit	*jEdit.org.gjt.sp.jedit.gui*	73	*AbbrevEditor.java*	15
JFreeChart	*jfreechart.source.org.jfree.chart.plot*	52	*AbstractPieLabelDistributor.java*	10
muCommander	*muCommander.main.com.mucommander.command*	15	*AssociationBuilder.java*	5

large project development experience. They are not familiar with the systems before. Then, they were assigned with a class as shown in the fourth column of Table 3. The task for them is to identify the most likely classes that are related to the given classes in their enclosing packages. Then each participant obtained a cluster of classes for each given class. As different participants may generate different clusters, they needed to discuss the results and reached a consensus on the clustering results for each given class. We used the clustering results as the authoritative clusters to compare with the clusters produced by our approach and the LSI-based/PLSA-based clustering approach. For LSI-based clustering approach and our approach that are used for comparison, we need to set the K value. Based on the size of the authoritative clusters, we set the K values for the packages, which are shown in the last column of Table 3. To answer *RQ2*, we first provided the clustering results of the three approaches to participants. In this process, to guarantee a fair treatment, they did not know which clustering results were generated by our approach or the LSI-based/PLSA-based clustering approach. Then, each of participants assessed each of the three clusters to vote the best one. In addition, to quantitatively compare these two approaches, we used precision and recall, two widely used information retrieval and classification metrics [41], to validate the accuracy of different clustering approaches. Precision measures the fraction of classes identified by a clustering approach to be in the same cluster as the given class that are truly relevant (based on the authoritative cluster), while recall measures the fraction of relevant results (i.e., classes that appear in the authoritative cluster) that are put in the same cluster as the given class by a clustering approach. Mathematically, they are defined as follows:

$$\begin{aligned} &\text{Precision} \\ &= \frac{\left|\text{Clustering results} \cap \text{Authoritative results}\right|}{\left|\text{Clustering results}\right|} \\ &\times 100\% \\ &\text{Recall} \\ &= \frac{\left|\text{Clustering results} \cap \text{Authoritative results}\right|}{\left|\text{Authoritative results}\right|} \\ &\times 100\%. \end{aligned} \tag{1}$$

In the above equations, clustering results and authoritative results are sets of classes.

To answer *RQ3*, participants were required to write the words in the identifiers or comments to label the authoritative clusters. This process is similar to that of *RQ2*, and a set of authoritative words are produced. To show whether the topics generated by our approach are useful, the participants needed to assess the generated topics to check whether they are useful for them to understand the clusters. Each participant needs to provide a rating in a five-point Likert scale, 1 (very useless) to 5 (very useful). Finally, we also computed the precision and recall of the words in the topics by comparing them with authoritative words. The way precision and recall are computed is similar to the way they are computed to answer *RQ2*.

Overall, the participants needed to answer four questions during the evaluation process. In *RQ2*, they were asked to give the answers of the authoritative results for the clustering and assess the results between *LSI* or *PLSI* and our approach. In *RQ3*, they needed to provide labels of the clusters and assess the topics generated by our approach.

5. Results

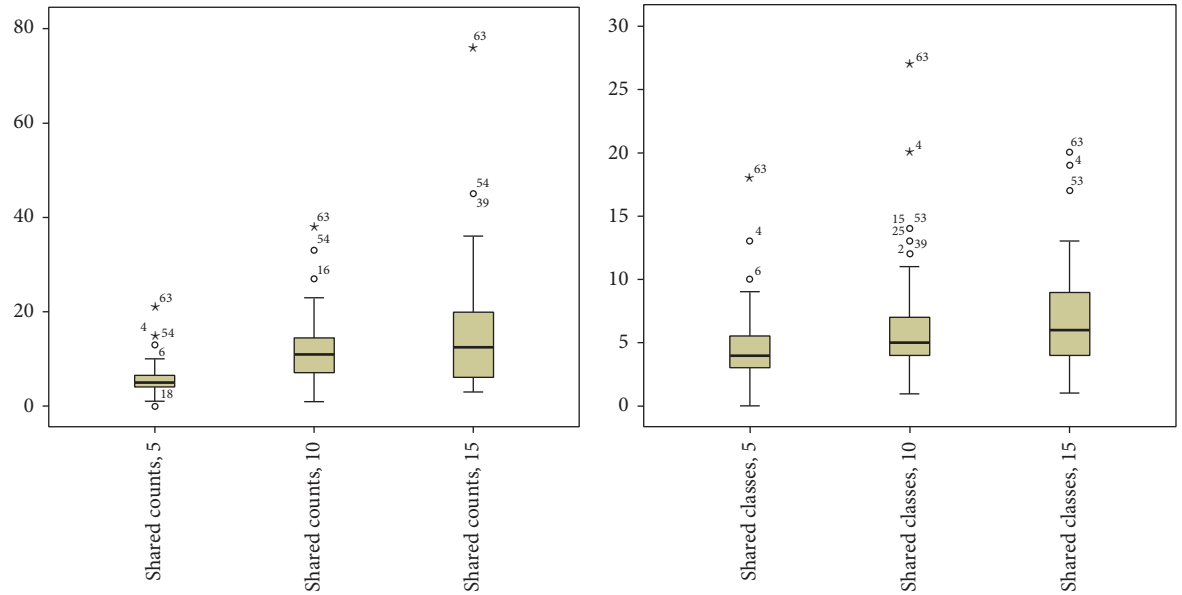
In this section, we gather and analyze results collected from the case studies to answer *RQ1*, *RQ2,* and *RQ3*.

5.1. RQ1. First, we see the existence of shared classes and nonmatching classes in the initial clusters. Table 4 shows the average percentage of the initial clusters without nonmatching classes and shared classes. From the results, we see that there do exist some shared classes and nonmatching classes in the initial clusters. So we need to perform the operation of reassigning these shared classes and nonmatching classes. Then, we see how the number of topics affects the results of shared classes and nonmatching classes. Figure 6 shows the box-plots of the number of shared classes and nonmatching classes and shared times in the process of clustering the classes with different numbers of topics. From the results in Figures 6(a) and 6(b), we notice that, with the increasing in the number of topics, the shared counts and the number of shared classes also increase. So setting different values of the number of topics will affect the number of shared classes. In addition, Figure 6(c) shows the results for nonmatching classes in the process of clustering the classes. We see that nonmatching classes are fewer than shared classes. Moreover, the range of the number of nonmatching classes with different numbers of topics is similar. That is, values of different numbers of topics do not obviously affect the number of nonmatching classes.

From the results discussed above, shared classes and nonmatching classes exist in the initial clusters. However, the majority of the classes are neither shared nor nonmatching

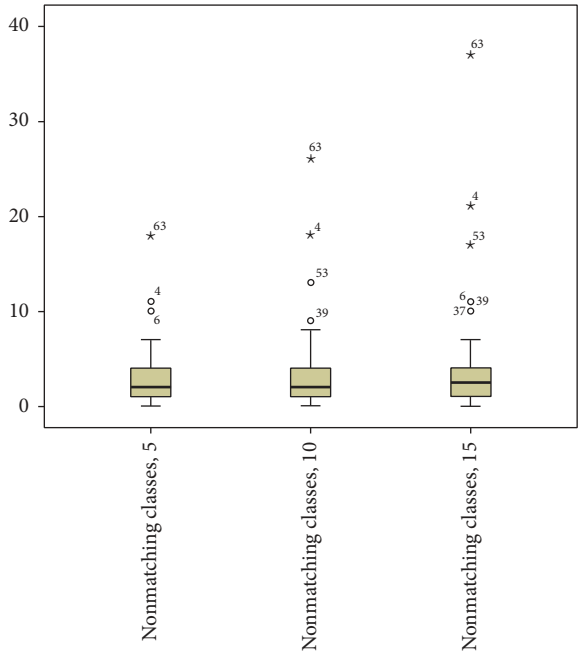
Table 4: The percentage of initial clusters without nonmatching classes and shared classes.

Cluster	$K = 5$ (%)	$K = 10$ (%)	$K = 15$ (%)
Initial clusters without nonmatching classes	90.2	90.1	85.4
Initial clusters without shared classes	70.0	60.0	55.3



(a) The shared counts of different classes in the initial clusters with different number of topics (5, 10, and 15)

(b) The number of shared classes in the initial clusters with different number of topics (5, 10, and 15)



(c) The number of nonmatching classes in the initial clusters with different number of topics (5, 10, and 15)

Figure 6: Shared counts, number of shared classes, and number of nonmatching classes in the initial clusters for number of topics set to 5, 10, and 15.

Table 5: The votes for our approach and LSI-based/PLSA-based clustering approach.

Subject	Package	Our approach	LSI-based	PLSA-based
JHotDraw	*JHotDraw.src.org.jhotdraw.app.action*	7	2	1
jEdit	*jEdit.org.gjt.sp.jedit.gui*	6	1	3
JFreeChart	*jfreechart.source.org.jfree.chart.plot*	5	4	1
muCommander	*muCommander.main.com.mucommander.command*	7	1	2

Table 6: The precision and recall of our approach and LSI-based/PLSA-based clustering approach.

Package	Our approach		LSI-based		PLSA-based	
	P	*R*	*P*	*R*	*P*	*R*
JHotDraw.src.org.jhotdraw.app.action	0.75	0.57	0.44	0.57	0.51	0.43
jEdit.org.gjt.sp.jedit.gui	0.4	0.5	0.04	0.25	0.14	0.22
jfreechart.source.org.jfree.chart.plot	0.83	0.83	1	0.6	0.76	0.68
muCommander.main.com.mucommander.command	0.67	0.33	0.29	0.33	0.42	0.26

ones. Furthermore, some shared classes are shared by three or more topics, and the number of shared classes is larger than that of nonmatching classes. In addition, the results also show that different settings of number of topics will affect the number of shared classes but will not affect the number of nonmatching classes.

5.2. RQ2. In this subsection, we compare the accuracies of the three clustering approaches to show the effectiveness of our approach.

First, we invited participants to assess the clustering results from three clustering approaches. The voting results are shown in Table 5. The results show that, in most cases, the results generated by our approach are more fit to their needs. For the *jfreechart.source.org.jfree.chart.plot* package, the voting results of LSI-based clustering and our approach are similar. When we investigated deep into the results in this package, both two clustering approaches output the clusters with two true-positive relevant classes (the true-positive relevant classes are those classes that do belong to the authoritative cluster). So participants are not sure which one is better than the other one. So from the participants' qualitative analysis, we notice that our approach can generate clustering results which better fit their needs compared to the LSI-based and PLSA-based clustering approach.

In addition, to quantitatively compare these clustering approaches, we compute their precision and recall results, which are shown in Table 6. From the recall perspective, our approach is always better than (or at least as good as) the LSI-based and PLSA-based clustering approaches. However, from the precision perspective, sometimes our results are better, and sometimes the *LSI-based clustering* or *PLSA-based clustering* is better. When we investigate results where our approach achieves lower precision, we notice that the number of classes in the cluster generated by our approach is larger than that by *LSI-based clustering* and *PLSA-based clustering*. For example, for *jfreechart.source.org.jfree.chart.plot* package, there are six classes in the authoritative cluster. Our approach generates five true-positive relevant classes while the *LSI-based clustering* approach generates four and the four are just the true-positive relevant classes. So the precision of the *LSI-based clustering* approach is high while our approach is worse. But from the respective of program comprehension, recall is more important because, with more relevant classes, developers can better comprehend the cluster. That is to say, our approach can cover more relevant classes in the authoritative clusters, which can effectively facilitate program comprehension. So from the results discussed above, compared with *LSI-based clustering* and *PLSA-based clustering*, our approach can effectively identify more relevant classes in a cluster to help program comprehension.

5.3. RQ3. Different from other clustering approaches, our approach also labels each cluster with the topics, which is composed of some words to describe the cluster. In this subsection, we discuss whether these topics are useful to comprehend the cluster.

First, we provided the topics of each cluster to the participants. They used a five-point Likert scale to answer the quality of the topics. The results are shown in Table 7. The average score of the results is around 4, which indicates that the participants think that the topics are useful to understand the cluster. So for program comprehension, the topics labeling the clusters are useful for users to understand the program.

In addition, we also assess the topics of the clusters quantitatively from the precision and recall perspectives. The results are shown in Table 8. For each cluster, our approach can produce a topic which includes some words to label it. These words can cover most of the words given by the participants. For example, for the cluster that includes the *AbstractPieLabelDistributor.java* class, 82% of the words can be covered. Hence, the participants can use these words to help them understand the clusters. In addition, from the precision perspective, our approach is not very good and most of the precision results are about 10%. However, for the other 90% irrelevant words, some are obviously not related

Table 7: The score assessed by the participants on the topics.

Package	Participants										
	P1	*P2*	*P3*	*P4*	*P5*	*P6*	*P7*	*P8*	*P9*	*P10*	*AVG*
JHotDraw.src.org.jhotdraw.app.action	5	5	5	5	4	5	5	4	4	4	4.6
jEdit.org.gjt.sp.jedit.gui	4	4	4	4	3	4	3	4	5	4	3.9
jfreechart.source.org.jfree.chart.plot	3	5	5	4	4	4	4	4	4	4	4.1
muCommander.main.com.mucommander.command	4	4	5	4	4	4	4	3	4	4	4

Table 8: The precision and recall of our approach in inferring representative words to label clusters.

Subject	Package	Class	Topics	
			P	*R*
JHotDraw	*JHotDraw.src.org.jhotdraw.app.action*	*AbstractProjectAction.java*	0.10	0.76
jEdit	*jEdit.org.gjt.sp.jedit.gui*	*AbbrevEditor.java*	0.06	0.80
JFreeChart	*jfreechart.source.org.jfree.chart.plot*	*AbstractPieLabelDistributor.java*	0.10	0.82
muCommander	*muCommander.main.com.mucommander.command*	*AssociationBuilder.java*	0.17	0.64

to the cluster, which are easily identified by the participants, for example, the words "*method,*" "*refer,*" and "*jEdit.*" These words are included in the topics because they are not removed in the preprocessing process. To improve the results, we can improve the preprocessing operations to remove words related to the specific subject programs. Although some noisy information is produced from our approach, the participants still feel that the topics are useful to understand the clusters.

Hence, from the results, we see that the topics in our approach are helpful for developers to understand the clustering results.

6. Threats to Validity

Like any empirical validation, ours has its limitations. In the following, threats to the validity of our case study are discussed.

The first threat relates to the correctness of our experiments and implementation. We have checked the implementation and fixed bugs. Another threat relates to participants' bias. We have reduced this bias by not telling the participants of results produced by our approach and those produced by the baseline approach. In addition, we only applied our technique to four subject programs. Moreover, we considered only one programming language (Java) and one development environment (Eclipse). Further studies are required to generalize our findings to large-scale industrial projects and with developers who have sufficient domain knowledge and familiarity with the subject systems. Thus we cannot guarantee that the results in our case study can be generalized to other more complex or arbitrary subjects. However, these subjects were selected from open-source projects and widely employed for experimental studies [42, 43]. In evaluating the effectiveness of the clustering results, we randomly selected a number of packages. To reduce the threats to validity further, in the future, we plan to evaluate our clustering approach with even more packages from more software projects. The final threat comes from the measures used to evaluate the effectiveness of our approach, that is, precision and recall. These two metrics only focused on the false-positives and false-negatives for authoritative clustering results. However, for program comprehension, other factors may be more important.

7. Related Work

Program comprehension is one of the most important activities in software maintenance and reverse engineering [8, 10, 23, 44, 45]. Clustering techniques are commonly used to decompose a software system into small units for easier comprehension. Some studies analyze syntax features or dependencies to cluster the software [46–50], while others rely on the semantic information in the source code for clustering [51–54].

Clustering approaches based on the syntax (structure) in the source code usually focus on the structural relationships among entities, for example, variable and class references, procedure calls, usage of packages, and association and inheritance relationships among classes. Mancoridis et al. proposed an approach which generates clusters using module dependency graph of the software system [8]. They treated clustering as an optimization problem, which makes use of traditional hill climbing and genetic algorithms. In [46, 55], the Bunch clustering system was introduced. Bunch generates clusters using weighted dependency graph for software maintenance. Sartipi and Kontogiannis presented an interactive approach composed of four phases to recover cohesive subsystems within *C* systems. In the first phase, relations between *C* programs are extracted. In the second phase, these relationships are used to build an attributed relational graph. In the third phase, the graph is manually or automatically partitioned using data mining techniques [56]. These syntax relationships can help developers understand how the functional features are programmed in the source code. In this article, we focus on the clustering based on functional features in the source code. And we used LDA for semantic analysis of these functional features.

Semantic based clustering approaches attempt to show the functional features of a system [57–60]. The functional features in the source code are analyzed from comments, identifier names, and file names [61]. Kuhn et al. presented a language independent approach to group software artifacts based on LSI. They grouped source code containing similar terms in the comments [12, 62]. Scanniello et al. presented an approach to perform the software system partitioning. This approach first analyzes software entities (e.g., programs or classes) and uses LSI to get the dissimilarity between entities, which are grouped by iteratively calling the *K-means* clustering algorithm [63]. Santos et al. used semantic clustering to support remodularization analysis in an input program [58]. Our approach used LDA to generate the clusters, particularly for large-sized packages, to facilitate their comprehension.

In addition, some program comprehension techniques combined the strengths of both syntax and semantic clustering [7, 38, 64–66]. The *ACDC* algorithm is one example of this combined approach which used name and dependency of classes to cluster all classes in a system into small clusters for comprehension [3]. Andritsos and Tzerpos proposed *LIMBO*, a hierarchical algorithm for software clustering [7]. The clustering algorithm considers both structural and nonstructural attributes to reduce the complexity of a software system by decomposing it into clusters. Saeidi et al. proposed to cluster a software system by incorporating knowledge from different viewpoints of the system, that is, knowledge embedded within the source code as well as the structural dependencies within the system, to produce a clustering result [67]. Then, they adopted a search-based approach to provide a multiview clustering of the software system. In this article, we focused on semantic analysis of the source code for its clustering. In addition, our approach also generates topics to help users more easily understand the classes in the clusters.

8. Conclusion and Future Work

In this article, we propose an approach of clustering classes in large-sized packages for program comprehension. Our approach uses LDA to cluster large-sized packages into small clusters, which are labeled with topics to show their features. We conducted case studies to show the effectiveness of our approach on four real-world open-source projects. The results show that the clustering results of our approach are more relevant than those of other clustering techniques, that is, LSI-based and PLSA-based clustering. In addition, the topics labeling these clusters are useful to help developers understand them. Therefore, our approach could provide an effective way for developers to understand large-sized packages quickly and accurately.

In our study, we only conducted studies on four Java-based programs, which does not imply its generality for other types of systems. Future work will focus on conducting more studies on different systems to evaluate the generality of our approach. In addition, during the clustering process, we find that some classes are weakly coupled with its package, but they are more related to another package. That is, there are problems with the current package's structure. So we consider applying our clustering approach to improve the package structure. Finally, our approach is a first step in a top-down program comprehension process; in the future, we plan to cluster other finer-level program elements, for example, methods, to provide a more comprehensive top-down program comprehension support to better understand a software system.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work is supported partially by Natural Science Foundation of China under Grant nos. 61402396, 61472344, 61602267, and 61472343, the Open Funds of State Key Laboratory for Novel Software Technology of Nanjing University under Grant no. KFKT2016B21, the Jiangsu Qin Lan Project, the China Postdoctoral Science Foundation under Grant no. 2015M571489, the Six Talent Peaks Project in Jiangsu Province under Grant no. 2011-DZXX-032, the Natural Science Foundation of the Jiangsu Higher Education Institutions of China under Grant no. 15KJB520030, the Priority Academic Program Development of Jiangsu Higher Education Institutions, and the Jiangsu Collaborative Innovation Center on Atmospheric Environment and Equipment Technology.

References

[1] X. Peng, Z. Xing, X. Tan, Y. Yu, and W. Zhao, "Improving feature location using structural similarity and iterative graph mapping," *Journal of Systems and Software*, vol. 86, no. 3, pp. 664–676, 2013.

[2] J. Wang, X. Peng, Z. Xing, and W. Zhao, "Improving feature location practice with multi-faceted interactive exploration," in *Proceedings of the 35th International Conference on Software Engineering (ICSE '13)*, pp. 762–771, IEEE, San Francisco, Calif, USA, May 2013.

[3] M. P. Obrien, "Software comprehension: a review and research direction," Tech. Rep., 2003.

[4] Z. Fu, K. Ren, J. Shu, X. Sun, and F. Huang, "Enabling personalized search over encrypted outsourced data with efficiency improvement," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 9, pp. 2546–2559, 2016.

[5] E. Soloway and K. Ehrlich, "Empirical studies of programming knowledge," *IEEE Transactions on Software Engineering*, vol. 10, no. 5, pp. 595–609, 1984.

[6] W. Maalej, R. Tiarks, T. Roehm, and R. Koschke, "On the comprehension of program comprehension," *ACM Transactions on Software Engineering and Methodology*, vol. 23, no. 4, article 31, 2014.

[7] P. Andritsos and V. Tzerpos, "Information-theoretic software clustering," *IEEE Transactions on Software Engineering*, vol. 31, no. 2, pp. 150–165, 2005.

[8] S. Mancoridis, B. S. Mitchell, C. Rorres, Y. Chen, and E. R. Gansner, "Using automatic clustering to produce high-level system organizations of source code," in *Proceedings of the*

6th International Workshop on Program Comprehension (IWPC '98), p. 45, Ischia, Italy, June 1998.

[9] N. Anquetil and T. Lethbridge, "Experiments with clustering as a software remodularization method," in *Proceedings of the 6th Working Conference on Reverse Engineering (WCRE '99)*, pp. 235–255, IEEE, October 1999.

[10] V. Rajlich and N. Wilde, "The role of concepts in program comprehension," in *Proceedings of the 10th International Workshop on Program Comprehension (IWPC '02)*, pp. 271–278, IEEE, Paris, France, June 2002.

[11] Z. Zhou, Y. Wang, Q. M. Wu, C. Yang, and X. Sun, "Effective and efficient global context verification for image copy detection," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 1, pp. 48–63, 2017.

[12] 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.

[13] S. C. Choi and W. Scacchi, "Extracting and restructuring the design of large systems," *IEEE Software*, vol. 7, no. 1, pp. 66–71, 1990.

[14] Y. S. Maarek, D. M. Berry, and G. E. Kaiser, "An information retrieval approach for automatically constructing software libraries," *IEEE Transactions on Software Engineering*, vol. 17, no. 8, pp. 800–813, 1991.

[15] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of Machine Learning Research*, vol. 3, no. 4-5, pp. 993–1022, 2003.

[16] X. Sun, X. Liu, B. Li, Y. Duan, H. Yang, and J. Hu, "Exploring topic models in software engineering data analysis: a survey," in *Proceedings of the 17th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD '16)*, pp. 357–362, IEEE, Shanghai, China, June 2016.

[17] B. Gu, V. S. Sheng, K. Y. Tay, W. Romano, and S. Li, "Incremental support vector learning for ordinal regression," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, no. 7, pp. 1403–1416, 2015.

[18] B. Gu, V. S. Sheng, Z. Wang, D. Ho, S. Osman, and S. Li, "Incremental learning for ν-support vector regression," *Neural Networks*, vol. 67, pp. 140–150, 2015.

[19] J. Tang, Z. Meng, X. Nguyen, Q. Mei, and M. Zhang, "Understanding the limiting factors of topic modeling via posterior contraction analysis," in *Proceedings of the 31th International Conference on Machine Learning*, pp. 190–198, 2014.

[20] A. Panichella, B. Dit, R. Oliveto, M. Di Penta, D. Poshynanyk, and A. De Lucia, "How to effectively use topic models for software engineering tasks? An approach based on genetic algorithms," in *Proceedings of the 35th International Conference on Software Engineering (ICSE '13)*, pp. 522–531, IEEE, May 2013.

[21] Y. Zhang, X. Sun, and B. Wang, "Efficient algorithm for k-barrier coverage based on integer linear programming," *China Communications*, vol. 13, no. 7, pp. 16–23, 2016.

[22] Q. Liu, W. Cai, J. Shen, Z. Fu, X. Liu, and N. Linge, "A speculative approach to spatial-temporal efficiency with multi-objective optimization in a heterogeneous cloud environment," *Security and Communication Networks*, vol. 9, no. 17, pp. 4002–4012, 2016.

[23] D. Binkley, D. Heinz, D. Lawrie, and J. Overfelt, "Understanding LDA in source code analysis," in *Proceedings of the 22nd International Conference on Program Comprehension (ICPC '14)*, pp. 26–36, June 2014.

[24] T. Mens, A. Serebrenik, and A. Cleve, Eds., *Evolving Software Systems*, Springer, 2014.

[25] F. Longo, R. Tiella, P. Tonella, and A. Villafiorita, "Measuring the impact of different categories of software evolution," in *Software Process and Product Measurement, International Conferences: IWSM 2008, Metrikon 2008, and Mensura 2008*, pp. 344–351, 2008.

[26] B. Dit, L. Guerrouj, D. Poshyvanyk, and G. Antoniol, "Can better identifier splitting techniques help feature location?" in *Proceedings of the IEEE 19th International Conference on Program Comprehension (ICPC '11)*, pp. 11–20, IEEE, Ontario, Canada, June 2011.

[27] T. Fritz, G. C. Murphy, E. Murphy-Hill, J. Ou, and E. Hill, "Degree-of-knowledge: modeling a developer's knowledge of code," *ACM Transactions on Software Engineering and Methodology*, vol. 23, no. 2, article 14, 2014.

[28] X. Sun, X. Liu, J. Hu, and J. Zhu, "Empirical studies on the NLP techniques for source code data preprocessing," in *Proceedings of the 3rd International Workshop on Evidential Assessment of Software Technologies (EAST '14)*, pp. 32–39, May 2014.

[29] G. Santos, M. T. Valente, and N. Anquetil, "Remodularization analysis using semantic clustering," in *Proceedings of the Software Evolution Week—IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE '14)*, pp. 224–233, Antwerp, Belgium, February 2014.

[30] T. Hofmann, "Unsupervised learning by probabilistic latent semantic analysis," *Machine Learning*, vol. 42, no. 1-2, pp. 177–196, 2001.

[31] T. Hofmann, "Probabilistic latent semantic analysis," in *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence (UAI '99)*, pp. 289–296, Stockholm, Sweden, July 1999, https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1&smnu=2&proceeding_id=15 &article_id=179.

[32] Y. Liu, D. Poshyvanyk, R. Ferenc, T. Gyimóthy, and N. Chrisochoides, "Modeling class cohesion as mixtures of latent topics," in *Proceedings of the IEEE International Conference on Software Maintenance (ICSM '09)*, pp. 233–242, Alberta, Canada, September 2009.

[33] M. Shtern and V. Tzerpos, "Clustering methodologies for software engineering," *Advances in Software Engineering*, vol. 2012, Article ID 792024, 18 pages, 2012.

[34] M. P. Robillard and G. C. Murphy, "Representing concerns in source code," *ACM Transactions on Software Engineering and Methodology*, vol. 16, no. 1, article 3, 2007.

[35] D. Binkley, M. Ceccato, M. Harman, F. Ricca, and P. Tonella, "Tool-supported refactoring of existing object-oriented code into aspects," *IEEE Transactions on Software Engineering*, vol. 32, no. 9, pp. 698–717, 2006.

[36] S. Deerwester, "Improving information retrieval with latent semantic indexing," in *Proceedings of the Annual Meeting of the American Society for Information Science*, pp. 1–10, 1988.

[37] 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, pp. 1–34, 2012.

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

[39] J. Han, *Data Mining: Concepts and Techniques*, Morgan Kaufmann, San Francisco, Calif, USA, 2005.

[40] X. Sun, B. Li, Y. Li, and Y. Chen, "What information in software historical repositories do we need to support software maintenance tasks? An approach based on topic model," in *Computer and Information Science*, pp. 27–37, Springer International Publishing, 2015.

[41] C. J. van Rijsbergen, *Information Retrieval*, Butterworths, London, UK, 1979.

[42] U. Erdemir, U. Tekin, and F. Buzluca, "Object oriented software clustering based on community structure," in *Proceedings of the 18th Asia Pacific Software Engineering Conference (APSEC '11)*, pp. 315–321, IEEE, Ho Chi Minh, Vietnam, December 2011.

[43] A. De Lucia, M. Di Penta, R. Oliveto, A. Panichella, and S. Panichella, "Using IR methods for labeling source code artifacts: is it worthwhile?" in *Proceedings of the 20th IEEE International Conference on Program Comprehension (ICPC '12)*, pp. 193–202, June 2012.

[44] X. Liu, X. Sun, B. Li, and J. Zhu, "PFN: a novel program feature network for program comprehension," in *Proceedings of the 13th IEEE/ACIS International Conference on Computer and Information Science (ICIS '14)*, pp. 349–354, Taiyuan, China, June 2014.

[45] Z. Fu, X. Wu, C. Guan, X. Sun, and K. Ren, "Toward efficient multi-keyword fuzzy search over encrypted outsourced data with accuracy improvement," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 12, pp. 2706–2716, 2016.

[46] B. S. Mitchell and S. Mancoridis, "On the automatic modularization of software systems using the bunch tool," *IEEE Transactions on Software Engineering*, vol. 32, no. 3, pp. 193–208, 2006.

[47] S. Islam, J. Krinke, D. Binkley, and M. Harman, "Coherent clusters in source code," *Journal of Systems and Software*, vol. 88, no. 1, pp. 1–24, 2014.

[48] S. Mirarab, A. Hassouna, and L. Tahvildari, "Using Bayesian belief networks to predict change propagation in software systems," in *Proceedings of the 15th IEEE International Conference on Program Comprehension (ICPC '07)*, pp. 177–186, June 2007.

[49] F. Deng and J. A. Jones, "Weighted system dependence graph," in *Proceedings of the 5th IEEE International Conference on Software Testing, Verification and Validation (ICST '12)*, pp. 380–389, Montreal, Canada, April 2012.

[50] M. Gethers, A. Aryani, and D. Poshyvanyk, "Combining conceptual and domain-based couplings to detect database and code dependencies," in *Proceedings of the IEEE 12th International Working Conference on Source Code Analysis and Manipulation (SCAM '12)*, pp. 144–153, IEEE, Trento, Italy, September 2012.

[51] Z. Fu, X. Sun, Q. Liu, L. Zhou, and J. Shu, "Achieving efficient cloud search services: multi-keyword ranked search over encrypted cloud data supporting parallel computing," *IEICE Transactions on Communications*, vol. E98B, no. 1, pp. 190–200, 2015.

[52] Z. Xia, X. Wang, X. Sun, and Q. Wang, "A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 2, pp. 340–352, 2016.

[53] L. Guerrouj, "Normalizing source code vocabulary to support program comprehension and software quality," in *Proceedings of the 35th International Conference on Software Engineering (ICSE '13)*, pp. 1385–1388, San Francisco, Calif, USA, May 2013.

[54] A. De Lucia, M. Di Penta, and R. Oliveto, "Improving source code lexicon via traceability and information retrieval," *IEEE Transactions on Software Engineering*, vol. 37, no. 2, pp. 205–227, 2011.

[55] N. Anquetil and T. C. Lethbridge, "Recovering software architecture from the names of source files," *Journal of Software Maintenance and Evolution*, vol. 11, no. 3, pp. 201–221, 1999.

[56] K. Sartipi and K. Kontogiannis, "A user-assisted approach to component clustering," *Journal of Software Maintenance and Evolution*, vol. 15, no. 4, pp. 265–295, 2003.

[57] T. Ma, J. Zhou, M. Tang et al., "Social network and tag sources based augmenting collaborative recommender system," *IEICE Transactions on Information and Systems*, vol. E98-D, no. 4, pp. 902–910, 2015.

[58] G. Santos, M. T. Valente, and N. Anquetil, "Remodularization analysis using semantic clustering," in *Proceedings of the 1st Software Evolution Week—IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE '14)*, pp. 224–233, February 2014.

[59] Z. Xia, X. Wang, X. Sun, and B. Wang, "Steganalysis of least significant bit matching using multi-order differences," *Security and Communication Networks*, vol. 7, no. 8, pp. 1283–1291, 2014.

[60] S. Kawaguchi, P. K. Garg, M. Matsushita, and K. Inoue, "Mudablue: an automatic categorization system for open source repositories," in *Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC '04)*, pp. 184–193, Busan, Republic of Korea, December 2004.

[61] A. Kuhn, S. Ducasse, and T. Gîrba, "Enriching reverse engineering with semantic clustering," in *Proceedings of the 12th Working Conference on Reverse Engineering (WCRE '05)*, pp. 133–142, Pittsburgh, Pa, USA, November 2005.

[62] A. Corazza, S. Di Martino, V. Maggio, and G. Scanniello, "Investigating the use of lexical information for software system clustering," in *Proceedings of the 15th European Conference on Software Maintenance and Reengineering (CSMR '11)*, pp. 35–44, IEEE, Oldenburg, Germany, March 2011.

[63] G. Scanniello, M. Risi, and G. Tortora, "Architecture recovery using Latent Semantic Indexing and k-Means: an empirical evaluation," in *Proceedings of the 8th IEEE International Conference on Software Engineering and Formal Methods (SEFM '10)*, pp. 103–112, September 2010.

[64] 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 *Proceedings of the 18th IEEE International Conference on Program Comprehension (ICPC '10)*, pp. 180–189, IEEE, Braga, Portugal, June-July 2010.

[65] G. Scanniello and A. Marcus, "Clustering support for static concept location in source code," in *Proceedings of the IEEE 19th International Conference on Program Comprehension (ICPC '11)*, pp. 36–40, Kingston, Canada, June 2011.

[66] Y. Kong, M. Zhang, and D. Ye, "A belief propagation-based method for task allocation in open and dynamic cloud environments," *Knowledge-Based Systems*, vol. 115, pp. 123–132, 2017.

[67] A. M. Saeidi, J. Hage, R. Khadka, and S. Jansen, "A search-based approach to multi-view clustering of software systems," in *Proceedings of the 22nd IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER '15)*, pp. 429–438, March 2015.